Creation of AES-Enabled Secure JAB-Code for Color Barcode Applications and its Integration with Medicinal Plants Information

Thesis

Submitted to the

G. B. Pant University of Agriculture & Technology
Pantnagar- 263145, Uttarakhand, India

By

Rishabh Kushawah
ID. No. 57979

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Engineering

(COMPUTER ENGINEERING)

August, 2023

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to my Advisor ***Prof. Jalaj Sharma*** *for providing me constant guidance and for being the pillar of support throughout this project. I would also like to acknowledge* ***Prof. B. K. Singh*** *and* ***Prof. C. S. Negi*** *for their knowledge and advices, which contribute significantly in my entire procedure of study. They always gave us suggestion that were crucial for making this project as flawless as possible.*

I would also like to thank ***Dr. Alaknanda Ashok,*** *Dean, College of Technology for their support and motivation. I would also like to thank* ***Dr. S. D. Samantaray****, Professor and Head, Computer Engineering Department for their constant support throughout the degree program.*

I sincerely admire the contribution of my batchmates Rajneesh, Akash, Meenal, Arshi, and Akanksha for their pleasant company, co-operation, help and for those memorable moments, which will always be remembered. I am also grateful to my junior Ekta for their pleasant company.

And I also thank to my friend Amit, Sourav and Akhil for their support in writing the thesis.

It is beyond my capacity to express my gratitude towards my parents, ***Dr. Raghu Nath Singh*** *and* ***Mrs. Shalini*** *and brother* ***Siddharth Kushawah*** *and, who are the pillars of my strength and have always stood to me to care, guide, encourage and unconditional support.*

Pantnagar
August, 2023

(Rishabh Kushawah)
Author

CERTIFICATE-I

This is to certify that the thesis entitled **"Creation of AES-Enabled Secure JAB-Code for Color Barcode Applications and its Integration with Medicinal Plants Information"** submitted in partial fulfillment of the requirements for the degree of **Master of Technology** with major in **Computer Engineering** of the College of Post Graduate Studies, G.B. Pant University of Agriculture and Technology, Pantnagar, is a record of *bonafide* research carried out by **Mr. Rishabh Kushawah ID. No 57979,** under my supervision and no part of the thesis has been submitted for any other degree or diploma.

The assistance and help received during the course of this investigation have been acknowledged.

Pantnagar
August, 2023

(Jalaj Sharma)
Chairperson
Advisory Committee

CERTIFICATE-II

We, the undersigned, members of the Advisory Committee of **Mr. Rishabh Kushawah ID. No 57979,** a candidate for the degree of **Master of Technology** with major in **Computer Engineering** agree that the thesis entitled **"Creation of AES-Enabled Secure JAB-Code for Color Barcode Applications and its Integration with Medicinal Plants Information"** may be submitted in partial fulfillment of the requirements for the degree.

(Jalaj Sharma)
Chairperson
Advisory Committee

(B. K. Singh)
Member

(C. S. Negi)
Member

CONTENTS

- **LIST OF TABLES**
- **LIST OF FIGURES**
- **LIST OF ABBREVIATIONS**

LIST OF TABLES

LIST OF FIGURES

LIST OFABBREVIATIONS USED

Abbreviations		Full Form
AES	:	Advanced Encryption
Standard UPC	:	Universal Product Code
ASCII	:	American Standard Code for Information interchnage
BMP	:	Bitmap
CBC	:	Cipher Block Chaining
CMYK	:	Cyan, Magenta, Yellow, and Key (Black)
CPU	:	Central Processing Unit
CQR	:	Color QR
DES	:	Data Encryption Standard
DES	:	Data Encryption Standard
ECC	:	Error Correction Code
GNOME	:	GNU Network Object Model Environment
GNU	:	GNU's Not Unix
GPU	:	Graphics Processing Unit
GUI	:	Graphical User Interface
HCCB	:	High-Capacity Color Barcode
HDD	:	Hard Disk Drive
ISO	:	International Organization for Standardization
JAB Code	:	Just Another Bar Code
JPEG	:	Joint Photographic Experts Group
LDPC	:	Low-Density Parity-Check
LGPL	:	GNU Lesser General Public License
LL	:	Lower Left
LR	:	Lower Right
LTS	:	Long-Term Support
LZ77	:	Lempel-Ziv 77
NIST	:	National Institute of Standards and Technology
OS	:	Operating System

PDF	:	Portable Document Format
PNG	:	Portable Network Graphics
QR	:	Quick Response
QR	:	Quick Response
RGB	:	Red Green Blue
SDK	:	Software Development Kit
SPN	:	Substitution-Permutation Network
SSD	:	Solid-State Drive
SVG	:	Scalable Vector Graphics
TIFF	:	Tagged Image File Format
UI	:	User Interface
UL	:	Upper Left
UR	:	Upper Right
VM	:	Virtual Machine

Introduction

Chapter 1 INTRODUCTION

The invention and development of barcodes has been one of the most impactful technological innovations of the twentieth century, reshaping numerous industries and becoming an integral part of our daily lives. From tracking inventory in warehouses to speeding up checkout processes in supermarkets and even facilitating ticketing in events or transport systems, barcodes have simplified and revolutionized data management systems. In the era of digital communication, data transmission has become a core aspect of many operations ranging from product authenticity verification to secure text transmission. Traditional barcode technology has played a pivotal role in such operations, with its application seen in numerous fields. However, conventional barcodes present several limitations such as limited data capacity and vulnerability to unauthorized access. This proposed method aims to overcome these limitations by introducing application: SecuJAB generator and SecuJAB reader that leverages Just Another Barcode (JAB Code), a high-capacity colored matrix barcode system coupled with Advanced Encryption Standard (AES) encryption for data security.

JAB Code, offers substantial benefits including higher data capacity and robustness against compared to black and white barcodes. However, its usage has been relatively limited due to the lack of user-friendly applications that can generate JAB Codes. Therefore, this method focuses on developing an application that bridges this gap, making JAB Code generation more accessible to users who are not necessarily technologically adept. To address the concern of data vulnerability inherent in barcode technology, the proposed application integrates AES encryption, a powerful data encryption standard. This ensures that the JAB Code can only be read by authorized persons thereby enhancing data security. The encryption process involves the user inputting a unique encryption key, which is required for decrypting the data encoded in the JAB Code. This key serves as an access control measure, allowing only authorized individuals to access the information. Given the growing need for secure and reliable means of verifying product authenticity, the proposed application could have significant implications in areas such as supply chain management and anti-counterfeiting efforts. By enabling the encoding of detailed product information into JAB Codes, the application can facilitate more efficient and reliable verification processes.

Moreover, the application can also be used for secure text transmission, providing a convenient and secure way for users to share sensitive information like question paper, personal details, patient identification, mark sheet details etc. By encoding the text data into a JAB Code, users can share this information confidently, knowing that it can only be accessed by those with the correct encryption key. In summary, this research presents an innovative solution to the challenges associated with traditional barcode technology, offering a more secure and user-friendly approach to data encoding and decoding through the integration of JAB Code and AES encryption.

1.1 History and Evolution of Barcodes

In 1974, the first Universal Product Code (UPC) was scanned on a packet of chewing gum at a supermarket in Troy, Ohio. This marked the commercial debut of barcodes. Since then, they have rapidly expanded in use and functionality. The barcoding technology evolved through the years, introducing different types of barcodes like Code 39, Code 128, and others. The 2D barcodes such as QR codes, Data Matrix, and PDF417 added another dimension to traditional barcodes, allowing them to store more information and be read from any direction.

1.2 Application of Barcodes in Today's Digital World

In today's digital world, barcodes play a crucial role in various sectors. They are used in logistics to track and manage goods; in retail to manage inventory and speed up checkouts; in healthcare for patient identification and tracking medication; and even in marketing where QR codes are used to quickly take a user to a website, video, or other online resources. The advancement of smartphones with barcode scanning capabilities has further expanded the use of barcodes. This research presents an application based on the latest development in barcoding technology, the JAB Code. It delves into the conceptualization, design, and implementation of this application, thereby contributing to the ever-growing and transformative field of barcoding technology.

1.3 Modern Barcoding Technology: JAB Code

The most recent advancement in this filed is the development of the Just Another Barcode (JAB) code. It's a 2D color barcode that is designed to store more information than its monochromatic counterparts. JAB Code, also known as ISO/IEC 24778, has the potential to store a vast amount of data in a small physical space. JAB Code uses 8 different colors to represent data, including black and white. This multicolored scheme allows for a higher data density. Given the same physical space, a JAB Code can store

more information than a conventional black and white barcode. This is particularly beneficial for industries and applications that need to encode a large amount of data in limited space.

1.4 JAB Code (Just Another Barcode)

JAB Code, short for Just Another Bar Code is an innovative two-dimensional color matrix symbology that enhances traditional barcode systems by adding a color dimension. By introducing color into the symbol, it allows for higher data density thereby providing the ability to encode larger amounts of data. The symbology is composed of basic symbols, which are colorful square modules organized in square or rectangular patterns, and it distinguishes between primary and secondary symbols. A JAB Code's structure includes one primary symbol, accompanied optionally by several secondary symbols, providing flexibility in data representation. The primary symbol is identifiable by four unique finder patterns located at its corners, this feature not present in secondary symbols. Secondary symbols can be docked to a primary symbol or another docked secondary symbol in either a horizontal or vertical direction, contributing to the versatility of the JAB Code. The encoding capacity of JAB Code ranges from small to large data sets, and it allows users to specify the level of error correction, ensuring the integrity of the encoded data. This specification further outlines the key attributes of the JAB Code symbology, such as its structural features, dimensions, cascading rules, data character encoding, error correction rules, and application parameters. The potential of JAB Code extends beyond the conventional use-cases of barcodes, offering promising opportunities for data representation and information management (**Berchtold *et al.* 2020)**.

1.4.1 Symbol Structure

1.4.1.1 Primary Symbol (Master Symbol)

The primary symbol of a JAB Code as represented in Fig. 1.1, embodies a unique structure encompassing various functional patterns. These include the finder pattern, alignment pattern, color palette, metadata, and encoded data region, all integral to the information storage and retrieval process in JAB Code. A defining feature of the primary symbol is the inclusion of four finder patterns strategically positioned at the four corners of the symbol. These finder patterns are separated from the border by a single module, thereby facilitating easier detection and decoding. Interestingly the JAB Code primary symbol doesn't necessitate a quiet zone around the symbol, which is a departure from traditional barcodes that often require a clear area surrounding the symbol to prevent scanner misreads.

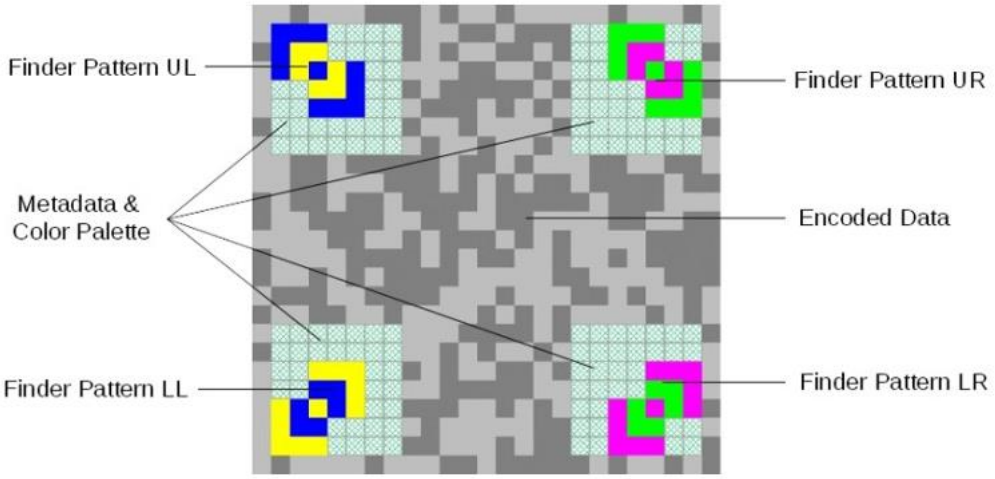

Fig. 1.1: Structure of a Primary Symbol (Master Symbol) (**Berchtold *et al.* 2020).**

1.4.1.2 Secondary Symbol (Slave Symbol)

The secondary symbol of a JAB Code in Fig. 1.2, exhibits a structure that is largely similar to the primary symbol with a few notable differences. Secondary symbols, like their primary counterparts, contain the same functional elements such as alignment patterns, metadata regions, and an encoded data region. However, the key distinction lies in the substitution of the four finder patterns - a characteristic feature of the primary symbol - with four alignment patterns in the secondary symbols. This alteration plays a critical role in distinguishing between primary and secondary symbols, while also maintaining the structural integrity of the overall JAB Code. Mirroring the primary symbol the secondary symbols do not necessitate a quiet zone in the surrounding area, making them more space-efficient compared to traditional barcodes.

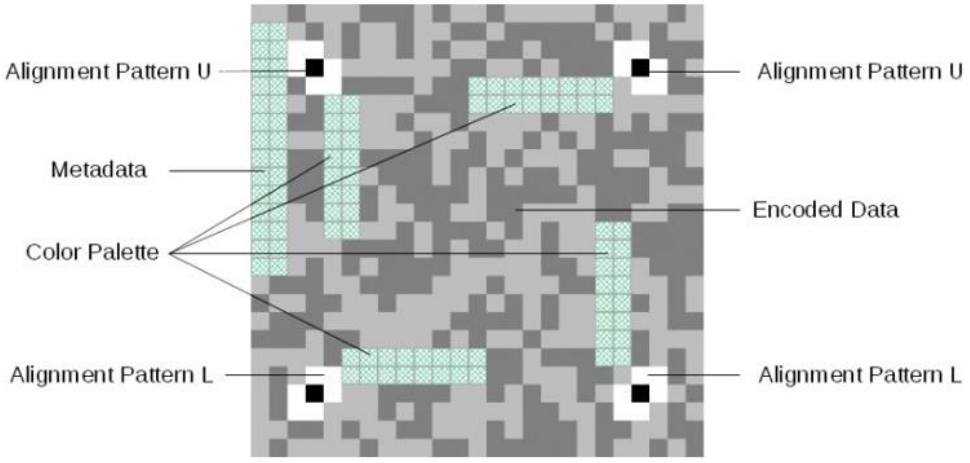

Fig. 1.2: Structure of a Secondary Symbol (Slave Symbol) (**Berchtold *et al.* 2020).**

1.4.1.3 Finder Pattern

In JAB Code symbology the concept of finder patterns is crucial, serving as key locational markers for decoding. As illustrated in Fig. 1.3, JAB Code utilizes four distinct types of finder patterns: Finder Pattern UL (Upper Left), UR (Upper Right), LR (Lower Right), and LL (Lower Left), positioned correspondingly at the upper left, upper right, lower right, and lower left corners of the primary symbol. Each finder pattern is constituted by two square references, each composed of a 3x3 module. These squares are connected by an overlapping module referred to as the 'core module', which plays a significant role in distinguishing the orientation of the finder patterns. In terms of orientation Finder Pattern UL and UR have their core module located at the lower right module of the upper reference and the upper left module of the lower reference, respectively. In contrast Finder Pattern LR and LL have the core at the lower left module of the upper reference and the upper right module of the lower reference.

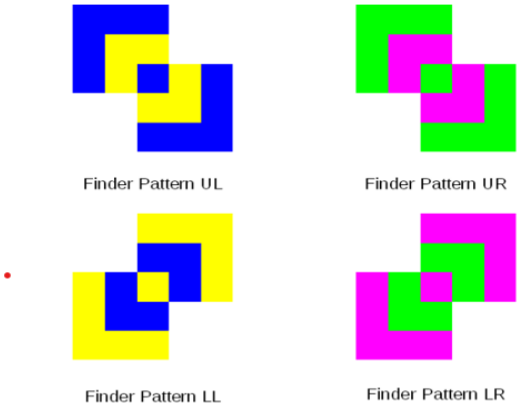

Fig. 1.3: Finder Patterns (**Berchtold *et al.* 2020).**

Each square reference in a finder pattern comprises three layers. These layers are symmetric around the core module. In a fascinating interplay of colors Finder Pattern UL and LL are composed of blue and yellow layers, while Finder Pattern UR and LR consist of green and magenta layers. Importantly each layer's color is distinct from its neighboring layers, and the core module for each type of finder pattern features a unique color. Specifically, Finder Pattern UL has a blue core, UR a green core, LR a magenta core, and LL a yellow core. These intricate details in the finder patterns of the JAB Code not only enhance its visual appeal but also contribute significantly to its functionality, enabling efficient and accurate decoding of encoded data.

1.4.1.4 Alignment Pattern

The alignment patterns in JAB Code play a crucial role in ensuring accurate positioning and decoding of data. As represented in Fig. 1.4, there are four types of alignment patterns in JAB Code, namely Alignment Pattern U, Alignment Pattern L, Alignment Pattern X0, and Alignment Pattern X1. Each alignment pattern is constructed of two square references, each consisting of 2x2 modules. These references are connected by an overlapping module known as the 'core module'. A distinct color scheme distinguishes Alignment Patterns U and L, which feature a white outer layer and a black core. Conversely, Alignment Patterns X0 and X1 display a black outer layer and a white core, clearly contrasting with the former two.

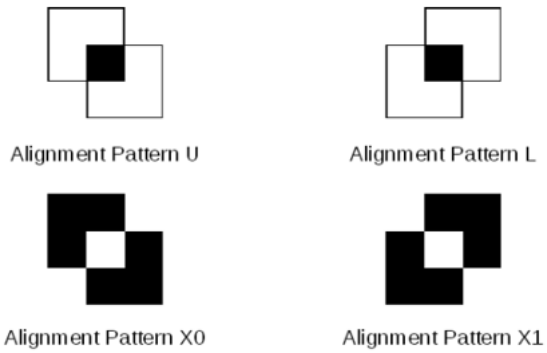

Fig. 1.4: Alignment Patterns (**Berchtold *et al.* 2020).**

The placement of these alignment patterns is carefully prescribed. Alignment Pattern U is to be located in secondary symbols at positions equivalent to those of Finder Pattern UL and UR in primary symbols. Similarly Alignment Pattern L is to be placed in secondary symbols at the same positions as Finder Pattern LR and LL in primary symbols. Alignment Patterns X0 and X1 are strategically placed between finder patterns in primary symbols and between Alignment Pattern U and L in secondary symbols. This placement strategy ensures efficient data decoding and maintains the overall structural coherence of JAB Code. These detailed characteristics and arrangements of alignment patterns contribute significantly to the robustness, reliability, and adaptability of the JAB Code system.

1.4.1.5 Color Palette

One of the most distinguishing features of JAB Code symbology is its color dimension, enabled by the implementation of a color palette. The color palette provides

reference module color values vital for accurate symbol decoding. As per Table 1.1, JAB Code supports 8 module color modes, thereby allowing the utilization of a minimum of 4 colors to a maximum of 8 colors in a symbol. This range illustrates the adaptability of JAB Code, with the color palette's size varying from 4 to a maximum of 8 (by modifying the code can goes up to 256 colors which is not iso certified). Table 1.1 illustrates 8-color palette.

Table 1.1: Color Palette of 8-Color Symbols (**Berchtold *et al.* 2020).**

Index	0	1	2	3	4	5	6	7
Color	Black	Blue	Cyan	White	Green	Yellow	Magenta	Red

This use of color palettes in JAB Code symbology not only enhances the aesthetic appeal but also significantly improves the information storage capacity, making it a versatile tool for diverse applications.

There are 2 modes available for color pallet 4 and 8 which are shown in Fig. 1.5 and 1.6.

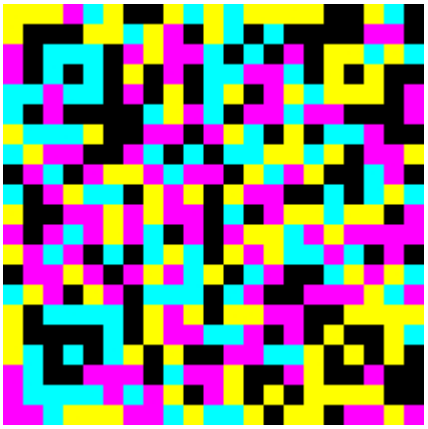

Fig. 1.5: 4-Color Mode

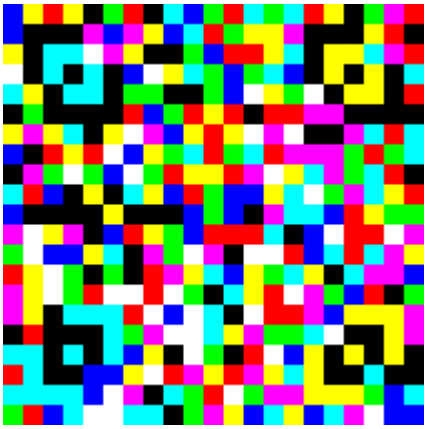

Fig. 1.6: 8-Color Mode

1.5 Advanced Encryption Standard (AES)

The American National Institute of Standards and Technology (NIST) developed the symmetric encryption algorithm known as the Advanced Encryption Standard (AES) in 2001. AES encrypts data in fixed-size blocks of 128 bits because it is a block cypher. AES supports key sizes of 128, 192, and 256 bits. Since AES is a repeated symmetric block cypher, the data is encrypted and decrypted using the same key. A substitution-permutation network (SPN), a design principle, serves as the foundation for AES. It consists of a number of interconnected processes, some of which require

changing inputs into particular outputs (substitutions), while others need moving bits about (permutations). AES uses numerous rounds of operation to transform plaintext into ciphertext. For 128-bit keys, there are 10 rounds; for 192-bit keys, there are 12 rounds; and for 256-bit keys, there are 14 rounds. The input plaintext is processed via a number of procedures in each round, including substitution, permutation, and mixing, to produce the final output of ciphertext. AES provides a high level of security, and it is efficient in both software and hardware including small devices. Its popularity also come from the fact that the algorithm is free to use and it does not have any known security issues when used properly. Because of this reason it has become the most popular algorithm for symmetric encryption.

The operation of AES encryption can be broken down into four key steps that are performed across multiple rounds. The overall process involves both the original plaintext block and the secret key. These steps are:

1. **SubBytes (Byte Substitution):** This is a non-linear substitution step where each byte in the block is replaced with another byte from a lookup table known as the S-box. The S-box used in AES is designed in such a way that there is no fixed relationship that can be exploited between the input and output bytes.
2. **ShiftRows (Row Shift):** In this step, bytes in each row of the block are shifted cyclically to the left. The top row is left unchanged, with each following row shifted one byte to the left more than the row above it. This serves to mix the data within each 128-bit block, which contributes to the security of the encryption.
3. **MixColumns (Column Mixing):** In the MixColumns step, the four bytes of each column in the block are combined using a linear transformation. This is done to further scramble the data and provide more security. This step is not performed in the final round of AES.
4. **AddRoundKey (Adding of the Round Key):** In the AddRoundKey step, the 128-bit block is bitwise XORed with the round key. The round keys are derived from the original encryption key using a key schedule.

This whole process is repeated for 10, 12, or 14 rounds, depending on the key length (128, 192, or 256 bits respectively).

The decryption process is the reverse of the encryption process, and uses the inverse of each step (InvSubBytes, InvShiftRows, InvMixColumns, AddRoundKey). However, the order of the rounds in the decryption process is opposite to the order in the encryption process. AES also includes an initial round of AddRoundKey before the main rounds start, and a final round that includes only the SubBytes, ShiftRows, and AddRoundKey steps (**Dworkin *et al.* 2001).**

AES is considered secure because it is computationally infeasible to derive the key from the output and because there is no known practical attack that can recover the key faster than a brute force search. Each operation in the AES algorithm contributes to this security by thoroughly scrambling the input data and the key. The design of the S-box and the MixColumns operation in particular provide resistance against linear and differential cryptanalysis, which are common methods for attacking block ciphers.

1.6 Motivation

In this era of technology-driven commerce expects the authenticity of products purchased either online or offline. The assurance of the product's authenticity is a fundamental right of every consumer. This motivation is driven by an incident where a consumer experienced the disappointment of purchasing counterfeit farming seeds. Despite the standard barcode verification on the packet the seeds turned out to be fraudulent. This event has underscored the pressing need for an advanced, solution that ensures the validity of products and secures information transmitted to consumers.

Subsequently, the increasing threat of data interception during transmission, coupled with the widespread of barcodes readable by anyone, prompted to create a secure text transmission mechanism that is easily accessible yet confidential to unauthorized persons.

This research, therefore, embarks on integrating the highly structured two-dimensional JAB Code with the powerful Advanced Encryption Standard (AES) to build a novel secure application. The application aims to serve two primary functions: firstly, to create JAB Codes, and secondly, to add security in JAB Code so only authorized person can read it.

1.7 Problem Statement

With the rise of digital technology, barcodes have become a ubiquitous tool for product identification and information storage. However, standard barcodes can be easily read by anyone, and the information embedded within them is not encrypted, posing a significant risk of data leakage. Additionally, the absence of a robust verification mechanism for product authenticity allows for the propagation of counterfeit products, damaging both the consumers and the legitimate manufacturers. Furthermore, in the contemporary digital landscape, there is a concerning absence of offline applications capable of creating and managing Just Another Bar Code (JAB Code), a recently developed two-dimensional barcode standard. This deficiency forces users to rely on potentially insecure online platforms for generating and handling JAB Codes, exposing their data to additional security threats.

Moreover, the need for a secure method of text transmission has been amplified with the increasing incidences of cyber-attacks and data breaches. While digital communication has undeniably expanded possibilities, it has also raised concerns about information security and privacy.

1.8 Objective

Based on these identified challenges, the primary objectives of this research are:

1) To create application for capable of creating JAB Codes offline as there is no offline application available for creating JAB Codes which forcing users to rely on potentially insecure online platforms.
2) To add security in JAB Code so only authorized person can read it. As the existing standard barcodes can be read by anyone which resulting in a lack of data privacy and security.
3) New verification method using application to counter issues of counterfeit products underlined by the incident of fraudulent farming seeds despite barcode verification.
4) Secured text transmission methods that can be easily accessed which mitigating the risk of unauthorized access to sensitive information.

The goal of this research is to develop an offline application that integrates the JAB Code system with the Advanced Encryption Standard (AES) to address these identified challenges. By doing so, it aims to enhance the security of text transmission, ensure that barcodes can only be read by authorized personnel, and provide a reliable tool for verifying product authenticity.

1.9 Thesis Outline

Chapter 1: This chapter describes the background information, motivation, problem statement, objective and significance of the study.

Chapter 2: This chapter is about review of literature, which is related to the work done in the field Color Barcode, application of barcodes and Research gap.

Chapter 3: This chapter explain the software, hardware, libraries and environments that have been used for the development of the application and methodology has been used for creating the application

Chapter 4: This chapter include result and discussion of the proposed work.

Chapter 5: This chapter deals with collusion, summary and future scope of the proposed work.

Review of Literature

Chapter 2 REVIEW OF LITERATURE

This chapter includes how various types of barcodes function and evolve based on past and recent research. Also examine the various barcode-based applications, focusing on how they increase efficiency and create new growth opportunities for various industries.

2.1 Previous Studies on Color Barcode and Traditional Barcode.

Berchtold *et al.* (2020) described a new variety of 2D barcode that encoded data using color and shape information. The JAB Code is intended to be extremely versatile and capable of data encoding. The JAB Code consist of numerous small barcodes arranged in a grid. Each small barcode, known as a JAB, contained color and shape data used to encode a portion of the overall data. The JAB Code is more robust than conventional 2D barcodes due to the utilization of multiple JABs. Redundancy and error-correction mechanisms increased the system's resistance to errors. The JAB Code's adaptability is one of its primary strengths. The color and shape information used during the encoding process could represent not only data but also metadata such as the language of the encoded text, the type of data contains, and contextual information. Additionally, the JAB Code is equipped with an error detection and correction mechanism based on Reed-Solomon error correction principles, which ensured greater precision and dependability. This made the JAB Code more resistant to noise and other forms of distortion that could impair the legibility of conventional 2D barcodes. The paper presented experimental results demonstrating the efficiency of the JAB Code for encoding and decoding a wide variety of data types. The authors also demonstrated that the JAB Code could be generated and decoded with ease using standard computer software. Overall, the JAB Code represented a promising new technology for encoding data in a flexible and secure manner, using color and shape information to provide greater redundancy and error correction than conventional 2D barcodes.

Winter *et al.* (2019) described a method for enhancing the security of physical documents through the use of digital signatures. The authors proposed using cryptographic digital signatures to generate more secure physical documents without the need for specialized hardware. The solution stored the digital signature and supplementary data on the document using JAB Code, a high-capacity matrix code.

This code is practical and economical because it was easily printable on the document. The proposed solution boasted several key features, one of which was offline verification. This feature allowed for the authenticity of a document to be verified without requiring an internet connection. Additionally, the solution supported long-term verifiability, enabling the authenticity of a document to be verified even after a considerable amount of time had passed since its issuance. The authors also defined a relaxed specification for short-term verification, ensuring that the verification process for such documents was not overly complicated. The paper demonstrated the proposed solution for a birth certificate and a prescription with a short expiration date. These were both essential documents that required strict security and verification measures. In conclusion, the proposed solution offered a promising method for enhancing document security without the need for specialized equipment. Using digital signatures and JAB Code to secure physical documents is efficient and cost-effective.

Mittra and Rakesh (2016) described a novel desktop application is which aims to enhance data security and authentication using QR codes. The core of this application lies in the modified Advanced Encryption Standard (AES) algorithm which is used to encrypt and decrypt data stored in QR codes. This differs from traditional QR code generation and scanning because it adds an extra layer of security to the data, making it more secure for transmission and storage. In order to generate a secure QR code, an ASCII matrix of the input data is created first. Then, using the modified AES algorithm, this data is encrypted and then mixed with a multi-dimensional bit-matrix of the QR code. This generates a secure, encrypted QR code. The process is reversed to read the QR code, with the added step of decrypting the AES-encrypted data using the same key. In comparison to other encryption algorithms such as DES, Triple DES, and Blowfish, the modified AES algorithm was found to be the best suited for the purpose of this paper due to its higher security and complexity. However, the paper also mentions some risks involved with the project such as its complexity and reliance on specific technology. The application is expect to have a broad range of applications, including secure transmission of question papers, crime investigation data, and other sensitive credentials within organizations. The paper further stresses that with advancements in technology, this application could also be run on mobile devices, expanding its utility further. A comparative study of different encryption algorithms has been provided which gives insights about the key length, key searched per second,

block size, type of the cipher text, the level of security and complexity of each algorithm. There is also a risk analysis table, providing details about the likelihood of each risk and the strategies for mitigating these risks. Overall, the paper provides a comprehensive overview of the creation and utility of a secure, AES-based QR code desktop application, highlighting its superior security profile, its potential applications, and the challenges in its development and implementation.

Dey *et al.* (2013) proposed an innovative approach to authenticate digital documents. This method is focused on solving the challenges surrounding the security and authenticity of data. The system works by encoding the marks obtained by a candidate in an encrypted form inside a QR Code. This encryption is implemented using the TTJSA encryption algorithm. When the marks are encrypted, they are then inserted into the QR code, which is printed along with the original data of the mark sheet. The purpose of this is to prevent an intruder from changing the marks on the mark sheet as they would not know the encryption key. The marks can be retrieved from the QR code and decrypted using the TTJSA decryption algorithm, and these can then be cross-verified with the marks already on the mark sheet.

Yfantis *et al.* (2012) demonstrated how two-dimensional barcodes are revolutionizing digital education. By scanning a QR code with a smartphone or QR code reader, learners gain instant access to a wide variety of digital content, ranging from video lectures to supplementary readings. Interactive activities and real-time assessments, facilitated by QR codes, heighten learner engagement and enable immediate feedback. QR codes can also unlock immersive augmented reality experiences, enriching the understanding of complex topics. In the realm of personalized education, QR codes are employed to create unique learning pathways that cater to individual needs. Despite their benefits, implementing QR codes requires careful consideration of factors like mobile optimization of digital content, user instruction for QR code usage, and strict privacy and security measures. Thus, Vasileios Yfantis' exploration of QR codes presents them as a powerful tool for improving e-learning by bridging offline and online resources, enhancing accessibility, and offering a more interactive educational journey.

Querini *et al.* (2011) introduced a new variation of 2D barcode that stores data using color information. Traditional black-and-white barcodes have limited data storage capacity, but the use of color has enabled more data to be encoded in the same physical

space. The authors proposed Color QR Code (CQR), a type of color barcode that uses color information to encode data and based on the well-known QR Code. CQR codes are easily readable by mobile devices and can be scanned using the same software as traditional QR codes. The authors provided a C++ written software library that could generate and decode CQR codes and was optimized for integration with mobile phone applications. Experiments demonstrated that CQR codes could store substantially more information than conventional QR codes while still being readable by mobile devices. The authors demonstrated that CQR codes are noise-resistant and applicable in real-world scenarios. Overall, the paper introduced a promising technology for encoding data using mobile-readable color barcodes.

Shao *et al.* (2010) addressed the challenge of long encrypting times in traditional AES (Advanced Encryption Standard) algorithms that are not suitable for rapid encryption needs. The main premise of this paper is that the high-performance computing capabilities of Graphics Processing Units (GPUs) can be used to improve the efficiency of the AES algorithm. Shao's research aims to enhance the AES algorithm using the high-performance computing capability of GPUs and compares the performance of this enhanced algorithm with that of a CPU-based implementation. The results from their experiments showed a significant improvement in the computing speed of the AES encryption algorithm when based on GPU, demonstrating higher performance than the AES encryption algorithm based on CPU.

Bulan *et al.* (2009) delve into the captivating domain of color barcodes, revealing their full potential through the use of dot orientation and color separability techniques. This innovative approach culminates in the creation of high-capacity color barcodes that have the potential to revolutionize how to store and retrieve information. Research is distinguished by its meticulous attention to detail and inventive strategies for overcoming the limitations of traditional black-and-white barcodes. Barcodes have become an integral part of our daily existence, as they facilitate inventory management, product tracking, and more. As the need for data storage continues to grow, however, it has become increasingly apparent that the storage capacity of monochromatic barcodes is reaching its limit. Color barcodes enter into play at this point. Although not a novel idea, Bulan's work elevates this concept by augmenting information density via the combination of color and dot orientation. The paper investigates in depth the challenges encountered in the development of color barcodes and the inventive solutions proposed

to overcome them. The issue of color separability is cited as a principal obstacle in the paper. Research presents a distinct color space that improves the ability to distinguish between various tones so as to accurately differentiate colors. Consequently, this allows the generation of a higher number of color combinations, thereby increasing the overall capacity of the barcode. In addition, the concept of dot orientation as a method of information encoding is explored. By taking into consideration the orientation of dots within the barcode matrix, research method provides an additional layer for data storage, significantly increasing information density without compromising readability. The author's innovative strategy, which integrates color separability and dot orientation, sets the groundwork for the development of high-capacity color barcodes, which could have a significant impact on multiple industries, from retail to logistics.

Parikh and Jancke (2008) presented a method for localizing and segmenting a new type of 2D barcode that makes use of color information to store more data than standard black-and-white barcodes. The proposed barcode, High-Capacity Color Barcode (HCCB), encodes data using a combination of black and white modules and colored triangles. Compared to traditional 2D barcodes, the use of color permits a much greater data capacity. This research describes a computer vision-based method for locating and segmenting HCCB codes in images. Detecting the black-and-white modules and colored triangles, followed by the placement of a rectangle around the entire barcode, constitutes the method. The authors evaluate the efficacy of their method using an HCCB-coded image dataset. In spite of noise and occlusion, they demonstrated that their method can accurately localise and segment the codes. In addition, the paper compares the capacity of HCCB codes to that of other forms of barcodes. The authors demonstrated that HCCB codes can store substantially more information than conventional 2D barcodes, making them appropriate for a variety of applications. The paper presents an intriguing new technique for encoding data using color barcodes, as well as a method for accurately localising and segmenting these codes in images. Potential applications of the strategy include product packaging, managing inventories, and document monitoring.

Table 2.1: Summary of Research Works Discussed in this Chapter.

Author(s)	**Year**	**Explanation**
Berchtold *et al.*	2020	Proposed the JAB Code, a new variety of 2D barcode using color and shape information for data encoding. Showcased its robustness, adaptability and resistance to errors.
Winter *et al.*	2019	Introduced a method for enhancing the security of physical documents using cryptographic digital signatures and JAB Code, a high-capacity matrix code. The solution offers offline verification and long-term verifiability.
Mittra and Rakesh	2016	Described a novel desktop application that enhances data security and authentication using QR codes with a modified Advanced Encryption Standard (AES) algorithm. Showcased its superior security and potential applications.
Dey *et al.*	2013	Proposed a system that encodes marks obtained by a candidate in an encrypted form inside a QR Code using the TTJSA encryption algorithm to ensure the security and authenticity of data.
Yfantis *et al.*	2012	Demonstrated how QR codes are revolutionizing digital education by offering instant access to digital content, enhancing learner engagement, and facilitating real-time assessments.
Querini *et al.*	2011	Introduced Color QR Code (CQR), a color barcode that uses color information to encode data. Showcased CQR's storage capacity, mobile-readability, and noise-resistance.
Shao *et al.*	2010	Proposed to enhance the AES algorithm using the high-performance computing capability of GPUs to improve the efficiency of the AES algorithm and reduce long encrypting times.
Bulan *et al.*	2009	Explored the potential of high-capacity color barcodes by using dot orientation and color separability techniques. Addressed the challenges encountered in the development of color barcodes.
Devi Parikh	2008	localizing and segmenting a new type of 2D barcode that makes use of color information to store more data than standard black-and-white barcodes. The proposed barcode, High-Capacity Color Barcode (HCCB), encodes data using a combination of black and white modules and colored triangles compared to traditional 2D barcodes.

2.2 Research Gap

- One significant gap in the existing QR code application is its limited data density. Traditional QR codes can only accommodate a finite amount of information in their square format. The application developed by Partiksha Mittra is subject to this limitation.
- A second notable gap is in data security. Without an encryption and decryption function, there's a potential risk to data privacy during transmission in the existing QR code application.
- Thirdly, there is a lack of configuration flexibility in QR code applications. Users are constrained to the standard square format and don't have options to change the layout such as to a 'U Shape', 'Vertical', or 'Horizontal' layout.
- The fourth, and possibly the most profound gap, is that little research has been conducted on JAB Code and there are virtually no offline applications available to create JAB Codes

Additionally, the existing QR code application might not be designed to handle file data, leaving a gap in the ability to encrypt file contents before representing them in barcode format. Furthermore, QR codes come with certain limitations in error correction capacity, which could compromise the accuracy and reliability of data transmission. Finally, traditional QR codes are limited to black and white colors, which can reduce data density compared to barcodes that can use multiple colors for encoding data. These gaps in existing QR code technology present significant challenges that your application aims to address with the use of JAB Code and other innovative features.

Materials and Methods

Chapter 3 MATERIALS AND METHODS

This chapter states about the materials and methods which are being used for the work. This study's main objective is to develop a graphical user interface (GUI) application using tkinter that enables the creation, reading, and encryption/decryption of barcodes with data, so that only authorized users have access. Python is used to construct the application, and the tkinter library is utilised to create the Crypto.Cipher user interface. The AES module used for encryption and decryption, as well as the subprocess module used to execute the JAB code writer. This comprehensive strategy provides an efficient method for handling data encryption, decryption, and the creation, reading of Barcodes using the JAB code.

3.1 Materials:

This section deals with the brief information of the hardware and the software that has been used in the proposed work.

3.1.1 Hardware used

The following system hardware specifications were used to complete the suggested work.

- **Processor:** Intel(R) Core (TM) i3-8130U CPU @ 2.20GHz 2.21 GHz
- **RAM (Installed Memory):** 20.0 GB (19.9 GB usable)
- **System Type:** 64-bit operating system, x64-based processor
- **Storage –** 1TB SSD, 1TB HDD

3.1.2 Software used

The proposed work was done in the VMware Workstation in Window 11, Ubnuntu-14.04 LTS and Ubuntu-22.04 LTS OS are used. The programming language that was used Python and C The details of all the software used are discussed below:

3.1.2.1 VMware Workstation

A robust virtualization program called VMware Workstation was created by VMware, Inc. to let users run different operating systems (OS) concurrently on a single physical machine. VMware Workstation offers an effective and adaptable solution for varied computing demands by generating isolated virtual machines (VMs) on the host computer, each with its own OS and applications.

1. Features and Functionality

a. **Seamless Integration:** VMware Workstation supports a wide range of operating systems, including Windows, Linux, macOS, and more, making it simple to integrate with different operating systems.

b. **Snapshots:** Users have the option of taking a snapshot of their virtual machines (VMs) at any moment, enabling them to swiftly return to a previous state if necessary. This is especially helpful when testing and debugging.

c. **Cloning:** Cloning is a feature of VMware Workstation that enables users to duplicate virtual machines without having to repeatedly install the operating system and software.

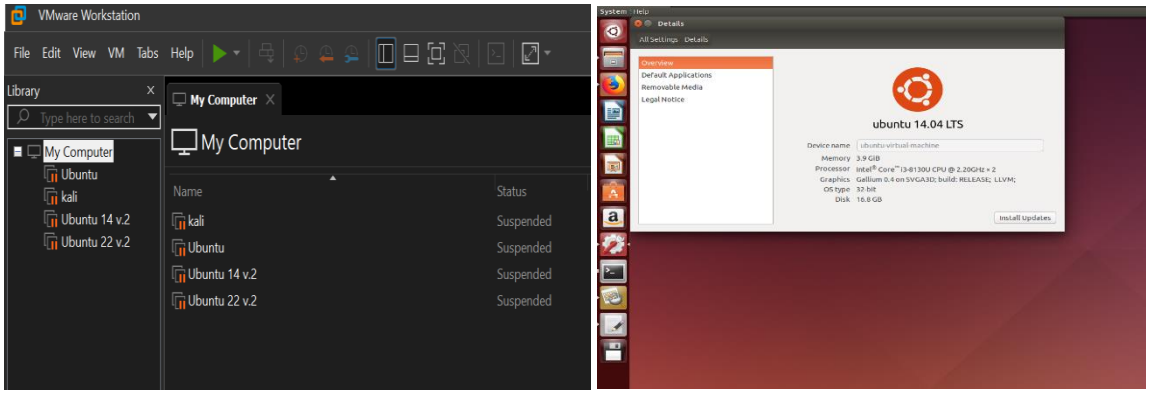

Fig. 3.1: VMware Workstation with Multiple OS

2. VMware Workstation Benefits

a. **Resource Efficiency:** By allowing several VMs to share the same physical resources, virtualization allows for improved resource utilization and reduces the need for extra hardware.

b. **Isolation:** Because VMs are isolated from one another, any problems or crashes within a single VM won't affect the host system or other VMs.

c. **Flexibility:** Users may quickly switch between several operating systems, use old software, or test out new programs in a safe setting without impacting their main system.

d. **Scalability:** User may easily add more virtual machines (VMs) to handle their rising computing demands without having to make substantial hardware investments.

3.1.2.2 Windows 11

Windows 11 is the latest operating system (OS) developed by Microsoft, succeeding Windows 10. Windows 11 brings a fresh, modern design and new features to enhance productivity, security, and user experience. VMware Workstation need Host OS to run the Virtual machine, Window 11 is used for that purpose in summary, Windows 11 is a significant update to Microsoft's operating system, focusing on a modernized design, improved productivity features, enhanced experiences, and stronger security. Its compatibility with a wide range of devices appeal to various user segments.

3.1.2.3 Ubuntu/Linux (14.04 LTS and 22.04 LTS)

Ubuntu is a Linux-based operating system derived from the Debian GNU/Linux distribution, offering open-source and free software that is easily accessible to all users. This versatile OS can function on both standalone computer systems and virtual machines, with support for Development features. Major advantage of Ubuntu is its plethora of libraries which help to make development easy. Recognized for its robust security, Ubuntu ensures users can run their applications safely. Essentially, Ubuntu is a contemporary, open-source operating system built on the Linux platform, serve to diverse user needs such as desktops, servers, IoT devices, and cloud computing.

1. Ubuntu 14.04 LTS

Ubuntu 14.04.6 LTS, also known as Trusty Tahr, is a Long-Term Support (LTS) release of the popular Linux-based operating system, Ubuntu. Launched in April 2014, this version received updates and support for five years, until April 2019. As an LTS release, it focused on providing a stable and secure foundation for users who prioritize reliability and long-term support over cutting-edge features. Trusty Tahr introduced several enhancements and improvements over its predecessors. Some notable features include a refined Unity desktop environment, improved hardware support, and better integration with online services. Additionally, Ubuntu 14.04.6 LTS incorporated updated software packages and security patches, ensuring an up-to-date and secure user experience. Ubuntu 14.04 is utilized for the initial testing of JAB Code due to its development being based on this version. The primary test conducted on this version provided valuable insights into the environment requirements and compatibility issues. The information gathered from this initial test aided in identifying libraries that were incompatible with newer operating systems and facilitated the search for suitable alternatives and resolutions to other potential issues.

2. Ubuntu 22.04 LTS

Ubuntu 22.04 LTS comes with range of updates and enhancements, including the Linux kernel 5.16, which offers new features, hardware compatibility, and bug fixes. The inclusion of Flutter SDK by default streamlines the development of native applications. Additionally, the latest GNOME 41 provides a more streamlined and simplified user interface (UI). With long-term support, Ubuntu 22.04 LTS has proven to be a reliable and stable version, delivering improved performance, a user-friendly interface, and robust security features. As a result of its widespread adoption and reliability, the JAB Code application was developed using this version.

3.1.2.4 Python

Guido van Rossum introduced Python, a versatile and interpreted programming language in 1991. As an open-source language, Python thrives on the support of large research community that persistently enhances its features and capabilities. With large number of packages designed for various applications, Python allows users to freely access and utilize these resources. Renowned for its user-friendly syntax and exceptional code readability. Python is widely employed for programming tasks and application development.

3.1.3 Medicinal Plants Images

Medicinal plants which are available at the university their images taken manually. Fig. 3.2 show the example of the images. By using images and JAB code together we are interested to provide following details.

1. **Common Name**: This is the name commonly used to refer to the plant in everyday language, as opposed to its scientific or botanical name.
2. **Botanical Name & Family**: The botanical name is the formal scientific name following the binomial nomenclature system, consisting of the genus and species of the plant. The family is a higher taxonomic rank in the hierarchy, grouping together related plants.
3. **Parts Used**: This column specifies which parts of the plant are used in medicinal or other applications. This can include leaves, roots, seeds, flowers, etc.
4. **Uses**: This describes the medicinal or other benefits of the plant. It may include its use as a treatment for specific health conditions, or other benefits.

Fig. 3.2: Medicinal Plant Images.

3.1.4 Dataset and its Description

Dataset is collected form university which have 250 medicinal plants details like Common Name, Botanical Name, Family, Parts user and Uses. Table 4.1 contains example description of dataset and its parameter.

Table 3.1: Dataset description for generation of JAB Code for Medicinal Plant.

S. No	Common Name	Botanical Name & Family	Parts used	Uses
1.	Atibala	*Abutilon indicum* Malvaceae	Leaves, root, seed	Nervine tonic, aphrodisiac, galactogogue, piles, diuretic
2.	Acalypha	*Acalypha indica* Euphorbiaceae	Whole plant	Skin disease, snakebite, toothache
3.	Achilia	*Achillea millefolium* Asteraceae	Leaves, flowers	Dysentry, fever, wound healing
4.	Apamarga	*Achyranthus aspera* Amaranthaceae	Root, plant	Stone, toothache, asthma, anemia, general debility
5.	Aloe	*Aloe barbadensis* Lilliaceae	Leaves juice, Roots	Skin disease, Jaundice, Joint pain Menstrual Problem.

3.2 Methodology

In this section, we will discuss the methodology and techniques involved in the proposed work. The process that has been used to achieve the proposed work is illustrated in the Fig. below.

3.2.1 JAB Code's Source Code

JAB Source Code is available on the GitHub repository and licensed under the GNU Lesser General Public License v2.1. Anyone can download the source code form - https://github.com/jabcode/jabcode, this source contains all the updates and changes of the source code that be done till now. The last source code updated according to ISO standardization, under https://www.iso.org/standard/76478.html, which specifies the characteristics of JAB Code symbology, symbol structure, symbol dimensions, symbol cascade rules, data character encoding, error correction rules, user-selectable parameters, print quality specifications, and a reference decode method. Under the LGPL-2.1 license following things can be done on the JAB Code.

Table 3.2: LGPL-2.1 License Terms

Permissions	Limitations	Conditions
Commercial use	Liability	License and copyright notice
Modification	Warranty	Disclose source
Distribution		State changes
Private use		Same license (library)

3.2.2 Libraries Required in Ubuntu 14.04

Based on the different OS, different libraries and other requirements need to be fulfilled, in this case libraries requirements for Compiling of the JAB Code Ubuntu 14.04 are used due to its development being based on this version (although this version is very old now and not in support). Which give the all the necessary information about the all the requirements, in this case following libraries are missing or not available in the OS.

1. -litff
2. -lpng16
3. -lz3
4. make

3.2.3 Compiling the Source Code

Build Instructions

1. Step 1: Execute the **make** command to create the JAB Code core library in **src/jabcode**
2. Step 2: Execute the **make** command to create the JAB Code writer in **src/jabcodeWriter**
3. Step 3: Execute the **make** command to create the JAB Code reader in **src/jabcodeReader**

The built library can be found in **src/jabcode/build**. The built reader and writer applications can be found in **src/jabcodeReader/bin** and **src/jabcodeWriter/bin**.

3.2.4 Testing the JAB Code

Execute the build file name "jabcodeWriter" on the terminal, will provide information output.

```
./JabcodeWriter
jabcodeWriter (Version 2.0.0 Build date: Nov 10 2022) - Fraunhofer SIT
Usage:
jabcodeWriter --input "message-to-encode" --output output-image [options]
--input              Input data (message to be encoded).
--input-file         Input data file.
--output             Output image file.
--color-number       Number of colors (4,8, default:8).
--module-size        Module size in pixel (default:12 pixels).
--symbol-width       Master symbol width in pixel.
--symbol-height      Master symbol height in pixel.
--symbol-number      Number of symbols (1-61, default:1).
--ecc-level          Error correction levels (1-10, default:3(6%)). If
                     different for each symbol, starting from master and
                     then slave symbols (ecc0 ecc1 ecc2...). For master
                     symbol, level 0 means using the default level, for
                     slaves, it means using the same level as its host.
--symbol-version     Side-Version of each symbol, starting from master and
                     then slave symbols (x0 y0 x1 y1 x2 y2...).
--symbol-position    Symbol positions (0-60), starting from master and
                     then slave symbols (p0 p1 p2...). Only required for
                     multi-symbol code.
--color-space        Color space of output image (0:RGB,1:CMYK,default:0).
                     RGB image is saved as PNG and CMYK image as TIFF.
--help               Print this help.
```

In output 2 example are provided for the understanding purpose, execution of the example is following below in Fig. 3.3 and Fig. 3.4.

Example for 1-symbol-code:

./jabcodeWriter --input 'Hello world' --output test.png

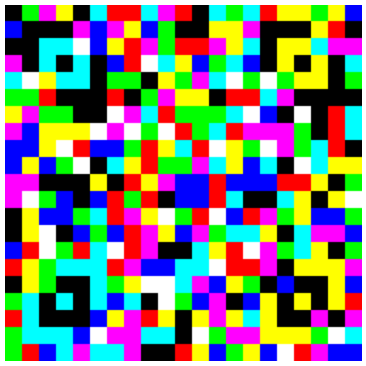

Fig. 3.3: Test normal JAB Code

Example for 3-symbol-code:

./jabcodeWriter --input 'Hello world' --output test.png --symbol-number 3 --symbol-position 0 3 2 --symbol-version 3 2 4 2 3 2

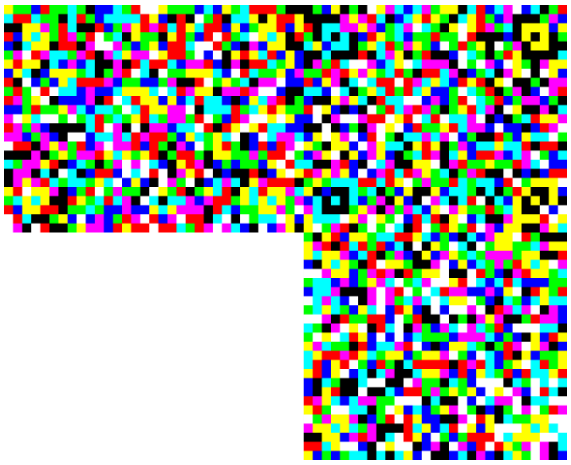

Fig. 3.4: Test multi symbol JAB Code

3.2.5 JAB Code's Architecture

Because JAB Code consist of one master and multiple salves, it can have multiple size and shape. Which required understating of the master and salve architecture, barcode always going to consist of at least one master (which is always be 0 in Fig. 3.5) or multiple slaves. Architecture of the master and slave is in the following Fig. 3.5.

					41					
				42	25	43				
			44	26	13	27	45			
		46	28	14	5	15	29	47		
	48	30	16	6	1	7	17	31	49	
59	39	23	11	3	**0**	4	12	24	40	60
	57	37	21	9	2	10	22	38	58	
		55	35	19	8	20	36	56		
			53	33	18	34	54			
				51	32	52				
					50					

Fig. 3.5: Order of Master and Slave Symbols

For Example, the Fig. 3.4 architecture will be following in Fig. 3.6.

Slave Position 3	Master Position 0
	Slave Position 2

Fig. 3.6: Positions of Master and Slave Symbol of Fig. 3.4

If user want to create the U Shape the following architecture will be used which is shown in the Fig. 3.7. This Figure consists of 4 slave positioning 6,3,7,4 and one master positioning 0.

Slave Position 6		Slave Position 7
Slave Position 3	Master Position 0	Slave Position 4

Fig. 3.7: Positions of Master and Slave Symbol of Fig. 3.8

By using this following U shape JAB Code will be generated.

Fig. 3.8: Test U Shape JAB Code

3.2.6 JAB Code's Arguments (Commands)

JAB Code work on multiple arguments, by combining the different arguments with the respected information a barcode can be generated. For making a reliable application which can create barcode need to gather all the necessary information, which are listed here.

a) **Input** - The text that needs to be encoded in the barcode (JAB Code), This only can be a string e.g. --input **'Hello world'**.

b) **input-file** - The data that needs to be encoded in the JAB Code. This can be a string, a file, or binary data, depending on the User's requirements. For this application only string and file option is provided for the users.

c) **Output** -The format and name of the generated JAB Code image (e.g., PNG,). This argument helps specify the desired output name and format for the JAB Code file "e.g. --output filename.png".

d) **color-number** - The color-number argument is a parameter that indicates how many colors will be applied in the barcode. By default, JAB code uses the 8 colors, The color-number input specifies the color scheme that will be used to represent the data modules in the master and slave symbols. JAB Code allows various colors. The number of colors directly affects the barcode's ability to store data because more colors allow the representation of more data in a smaller space.

There are only two options are allowed by the ISO standards.

1. 4-color mode, blue, green, magenta and yellow only be used.
2. 8-color mode, black, green, cyan, blue, magenta, red, yellow and white.

When testing is being done on the JAB Code by tinkering on the code other colors mode 16,32,64,128,256 can be enable (which are not ISO certified).

e) **module-size** - The module-size argument refers to the size of the individual modules (data cells) within the barcode. Modules are the smallest units in a JAB Code barcode, arranged in rows and columns to form the master and slave symbols. It determines the physical size of each module in the generated barcode image, large module size make large barcode and smaller module size make small barcode while the capacity will be going to be same for both same barcodes. The choice of module size can be considered with following factors.

   1) **Print resolution:** A higher print resolution can accommodate smaller module sizes, while lower resolution printers might require larger module sizes to ensure accurate printing and scanning.
   2) **Scanning equipment:** More advanced scanning equipment can reliably read smaller module sizes, while basic scanning devices might require larger modules for better readability.
   3) **Space constraints:** smaller module sizes can be used when space is limited, while larger module sizes can improve readability and error correction at the expense of increased barcode size.

f) **symbol-width** - The symbol-width argument refers to the width of the master and slave symbols in terms of the number of modules (data cells) per row. This parameter affects the overall layout and appearance of the JAB Code barcode. When encoding data into a JAB Code barcode, the symbol-width argument specifies the number of modules per row for the master symbol and, if applicable, the slave symbols. The symbol width, along with the symbol height (i.e., the number of modules per column), determines the overall size and shape of the barcode. barcode with larger symbol width will result in a wider barcode with a potentially higher data capacity, while a smaller symbol width will produce a more compact barcode but might accommodate less data.

g) symbol-height - The symbol-height argument refers to the height of the master and slave symbols in terms of the number of modules (data cells) per column. This parameter impacts the overall layout and appearance of the JAB Code barcode, just like the symbol-width argument. When encoding data into a JAB Code barcode, the symbol-height argument specifies the number of modules per column for the master symbol and, if applicable, the slave symbols. The symbol height, along with the symbol width (i.e., the number of modules per row), determines the overall size and shape of the barcode. With larger symbol height will result in a taller barcode with a potentially higher data capacity, while a smaller symbol height will produce a more compact barcode but might accommodate less data.

The choice of symbol height and width should consider factors such as:

1) **Aspect ratio:** The symbol height and width should be chosen to produce a barcode with a suitable aspect ratio for the intended application, print media, or scanning equipment.

2) **Data capacity:** larger symbol dimensions can accommodate more data, but they may increase the overall size of the barcode and might require more sophisticated scanning equipment to read accurately.

3) **Print and scanning constraints:** The symbol height and width should be chosen with the specific print and scanning equipment in mind, ensuring that the barcode is legible and can be accurately decoded.

h) symbol-number - The symbol-number argument refers to the total number of symbols (including the master symbol and all slave symbols) in the barcode. This parameter impacts the overall data capacity and layout of the JAB Code barcode. It specifies how many symbols (master and slaves) are needed to accommodate the data. The number of symbols directly impacts the data capacity and structure of the barcode. A higher symbol-number value will result in a larger overall barcode with a higher data capacity. However, it may require more sophisticated scanning equipment to read accurately. Conversely, a lower symbol-number value will produce a smaller and more compact barcode but might accommodate less data.

1) **Master Symbol:** There is always one master symbol that contains crucial information required to decode the entire barcode, such as size, color palette, and metadata. The master symbol is typically positioned in the top-left corner and must be scanned first.

2) **Slave Symbols:** The number of slave symbols depends on the amount of data to be encoded and the symbol-number argument. Slave symbols store the actual data payload and are linked to the master symbol. They can be arranged in a grid-like pattern around the master symbol, and their placement is determined by the metadata within the master symbol.

i) ecc-level - The ecc-level (Error Correction Code level) argument is a parameter that specifies the level of error correction to be applied during the encoding process. Error correction is a mechanism that allows the recovery of encoded data even in cases of damage, distortion, or partial obstruction of the barcode shown in table 3.1. JAB Code uses Reed-Solomon error correction codes to achieve its robustness. The ecc-level determines the amount of redundant error correction information added to the barcode. There are typically several levels of error correction available, with higher levels providing better resilience against damage but at the expense of increased barcode size and reduced data capacity.

The ecc-level should follow factors such as:

1) **Barcode Durability:** If the barcode is expected to be exposed to wear, tear, or damage (e.g., on product packaging or in harsh environments), a higher ecc-level may be necessary to ensure reliable decoding.

2) **Scanning Accuracy:** In situations where scanning equipment may be less accurate or where the barcode may be partially obstructed, a higher ecc-level can improve the chances of successful decoding.

3) **Data Capacity:** Higher ecc-levels result in larger barcodes with reduced data capacity, as more space is dedicated to error correction information. The ecc-level should be chosen to strike a balance between error correction robustness and data capacity.

Table 3.3: The Approximated Amount of Bit Error Recovery Capacity in % (**Berchtold *et al.* 2020).**

Level	Recovery capacity in %
0	3
1	4
2	5
3	6
4	7
5	8
6	9
7	10
8	11
9	12
10	14

j) symbol-version - The symbol-version argument specifies the dimensions (width and height) of each symbol in the barcode.

For example, in Fig. 3.4

--symbol-version 3 2 4 2 3 2: argument specifies the dimensions (width and height) of each symbol in the barcode. The master symbol has a width of 3 and a height of 2. The first slave symbol has a width of 4 and a height of 2. The second slave symbol has a width of 3 and a height of 2. Side-Version of each symbol, starting from master and then slave symbols (x0 y0 x1 y1 x2 y2...).

k) symbol-position – It defines position information for slave symbols relative to the master symbol. For example, in Fig. 3.5

--symbol-position 0 3 2: The position information for slave symbols relative to the master symbol. In this case, there are two slave symbols: the first slave symbol is positioned 0 columns to the right and 3 rows below the master symbol, and the second slave symbol is positioned 2 columns to the right and 3 rows below the master symbol. Symbol positions (0-60), starting from master and then slave symbols (p0 p1 p2...). Only required for multi-symbol code

l) **color-space** – JAB Code typically uses the CMYK color model (Cyan, Magenta, Yellow, and Key/Black) to define colors. However, other color models, such as RGB (Red, Green, Blue), could be used. Color space of output image (0: RGB,1: CMYK, default:0). RGB image is saved as PNG and CMYK image as TIFF.

m) **help** - Print for help information.

3.2.7 Requirements for Ubuntu 22.04 LTS

Ubuntu 22.04 LTS is used for new OS, while setting up the JAB Code following error and missing libraries were found (Error may differ on different OS and versions).

1. **libtiff-dev** (**Tag Image File Format library) –** For Ubuntu Linux, the TIFF (Tagged Image File Format) library has a development package called libtiff-dev. Which Support for reading, writing, and modifying TIFF images, provided by the TIFF library. It's commonly used in various software applications that deal with images, such as image editors, viewers, or converters.

   To install libtiff-dev package (library) on Ubuntu, run the following command on terminal:

   ```
   sudo apt-get update
   sudo apt-get install libtiff-dev
   ```

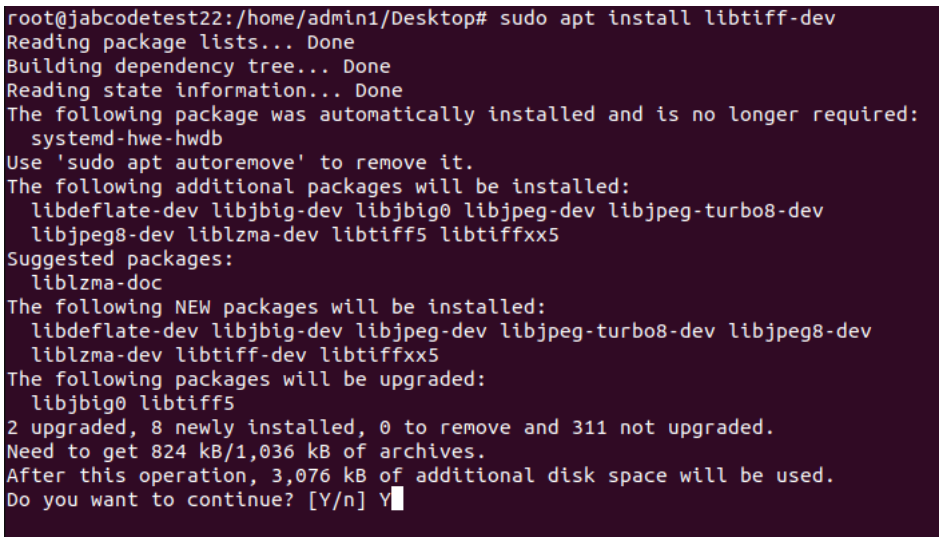

Fig. 3.9: Installing "libtiff-dev" library

This will first update the package list on present system and then install the libtiff-dev package, along with any required dependencies. when library installed, user can use the library in development projects or compile software that depends on the TIFF library.

2. **libpng-dev (PNG (Portable Network Graphics) Library) -** For Ubuntu Linux, the PNG (Portable Network Graphics) library has a development package called libpng-dev. The widely used lossless picture format known as PNG which supported for reading, writing, and manipulation by the PNG library. It is commonly used in various software applications, such as image editors, viewers, or converters.

   To install libpng-dev package (library) on Ubuntu, run the following command on terminal:

   ```
   sudo apt-get update
   sudo apt-get install libpng-dev
   ```

   This will install libpng library but this version is not supported by the JAB Code which was build on the older version of the libpng library, for overcoming this error older version of libpng is utilized.

   Following step need to performed for the older version of libpng.

   1. Download https://sourceforge.net/projects/libpng/files/libpng16/older-releases/1.6.32/
   2. Extract the file
   3. Open terminal on extracted file location
   4. Execute the following command on terminal.
      a. ./conFig.
      b. make clean
      c. make check
      d. sudo make install

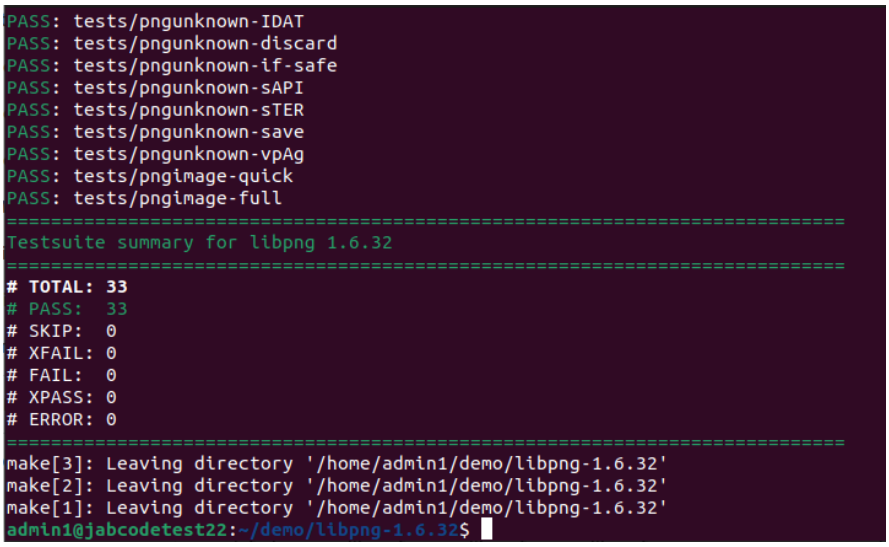

Fig. 3.10: Checking if "libpng" Library can be Installed.

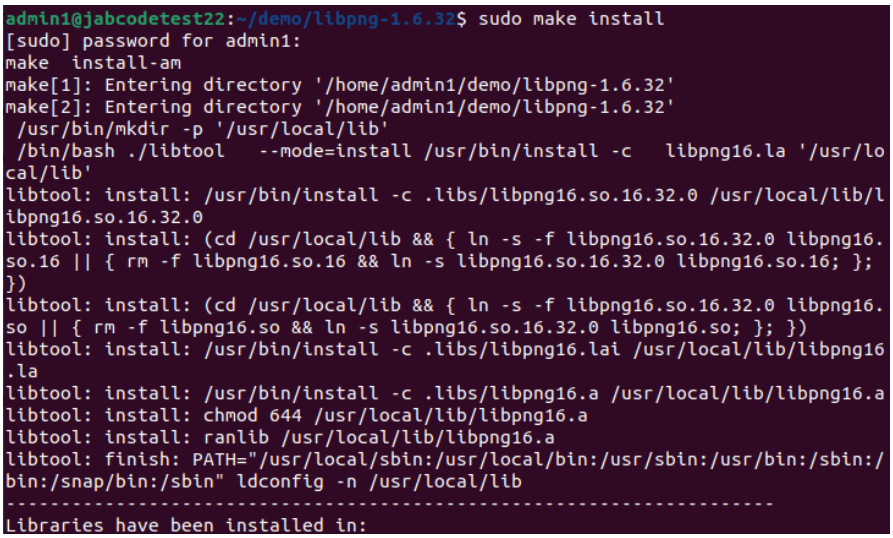

Fig. 3.11: Installed "libpng" Library.

This will first update the package list on present system and then install the libpng package, along with any required dependencies. when library installed, user can use the library in development projects or compile software that depends on the PNG library.

3. **jabcodeWriter Error Solution –** In the source code of jabcodeWriter folder which consist of the dependencies and instruction for program how to execute the compilation, in that instruction File Name – "Makefile" has wrong instruction which is stopping the execution.

   Instruction which is creating error – "**-L../jabcode/lib**"

This instruction telling compiler to take library form "jabcode/lib" folder at the time of compiling the program which is consist of 4 libraries **-ltiff, -lpng16, -lz and -lm** which are not usable for the current OS. To prevent that error this instruction need to be removed from the file and saved as it shown in Fig. 3.12.

```
1 PREFIX  =
2 CC      = $(PREFIX)gcc
3 CFLAGS   = -O2 -std=c11
4
5 TARGET = bin/jabcodeWriter
6
7 OBJECTS = $(patsubst %.c,%.o,$(wildcard *.c))
8
9 $(TARGET): $(OBJECTS)
10        $(CC) $^ -L../jabcode/build -ljabcode -L../jabcode/lib -ltiff -lpng16 -lz -lm $(CFLAGS)
  -o $@
11
12 $(OBJECTS): %.o: %.c
13        $(CC) -c -I. -I../jabcode -I../jabcode/include $(CFLAGS) $< -o $@
14
15 clean:
16        rm -f $(TARGET) $(OBJECTS)
```

Fig. 3.12: Error in "jabcodeWriter" File

Now compile the jabcodeWriter folder, which will give the following output in Fig. 3.13.

```
admin1@jabcodetest22:~/Downloads/jabcode-master/src/jabcodeWriter$ make
gcc jabwriter.o -L../jabcode/build -ljabcode -ltiff -lpng16 -lz -lm -O2 -std=c11 -o bin/jabcode
Writer
admin1@jabcodetest22:~/Downloads/jabcode-master/src/jabcodeWriter$ make
make: 'bin/jabcodeWriter' is up to date.
admin1@jabcodetest22:~/Downloads/jabcode-master/src/jabcodeWriter$
```

Fig. 3.13: Compiled "jabcodeWriter" File after Removing Error

4. **jabcodeReader Error Solution -** In the source code of jabcodeReader folder which consist of the dependencies and instruction for program how to execute the compilation, in that instruction File Name – "Makefile" has wrong instruction which is stopping the execution.

   Instruction which is creating error – "**-L../jabcode/lib**"

   This instruction telling compiler to take library form "jabcode/lib" folder at the time of compiling the program which is consist of 4 libraries -ltiff, -lpng16, -lz and -lm which are not usable for the current OS. To prevent that error this instruction need to be removed from the file and saved as it shown in Fig. 3.14.

```
1 PREFIX  =
2 CC      = $(PREFIX)gcc
3 CFLAGS   = -O2 -std=c11
4
5 TARGET = bin/jabcodeReader
6
7 OBJECTS = $(patsubst %.c,%.o,$(wildcard *.c))
8
9 $(TARGET): $(OBJECTS)
10        $(CC) $^ -L../jabcode/build -ljabcode -L../jabcode/lib -ltiff -lpng16 -lz -lm $(CFLAGS)
  -o $@
11
12 $(OBJECTS): %.o: %.c
13        $(CC) -c -I. -I../jabcode -I../jabcode/include $(CFLAGS) $< -o $@
14
15 clean:
16        rm -f $(TARGET) $(OBJECTS)
```

Fig. 3.14: Error in "jabcodeReader" File

Now compile the jabcodeReader folder, which will give the following output in Fig. 3.15.

```
admin1@jabcodetest22:~/Downloads/jabcode-master/src/jabcodeReader$ make
gcc jabreader.o -L../jabcode/build -ljabcode -ltiff -lpng16 -lz -lm -O2 -std=c11 -o bin/jabcode
Reader
admin1@jabcodetest22:~/Downloads/jabcode-master/src/jabcodeReader$ make
make: 'bin/jabcodeReader' is up to date.
```

Fig. 3.15: Compiled "jabcodeReader" File after Removing Error

By solving all the error jabcodeWriter and jabcodeReader will be compiled and can be for the further process.

3.2.8 Testing of JAB Code in Ubuntu 22.04 LTS

Now jabcodeWriter and jabcodeReader can be executed, and testing of these programs can be done (whether to check if the program is compiled successfully or not). This testing is need to be done on terminal, for each testing separate argument need to be created and pass it threw the program and by comparing with the desired output test is validated. Multiple tests been performed for both jabcodeWriter and jabcodeReader

3.2.9 Requirement Analysis for JAB Code Application

An important element in the development process of application is requirements analysis, that helps in identifying and prioritising the features and functionalities which are required for creating a successful application. Requirements can be broadly categorized into following.

1. **Functional Requirements:**
   a) **Data Input:** The application able to support both text and file input (text file consist of data). Users should be able to type or paste text into a dedicated input field or select a file from their system to be encoded into a JAB Code.

b) **Encryption:** The application should allow users to enable or disable AES encryption for input data. If encryption is enabled, users need to provide a valid encryption key with a length that is 128/192/256 bits. The application should handle encryption and decryption using the provided key.

c) **JAB Code Generation:** Implement a JAB Code generation process that takes the input data (text or file), optional encryption, and selected symbol layout to generate a JAB Code image in PNG format. This process should utilize the JAB Code writer executable or a compatible library, and show the Barcode output to the user when it's generated.

d) **Error Handling:** The application should be able to handle common errors very easily, such as invalid encryption keys, missing input data, or issues with the JAB Code writer executable. It should display meaningful error messages to the user when the error occurs.

e) **Symbol layout selection:** The application should offer multiple JAB Code symbol layout options (None, U Shape, Vertical, Horizontal) and allow users to choose their desired layout from a dropdown menu.

2. **Non-functional Requirements:**

   a) **User Interface (UI):** Development of graphical user interface (GUI) must be user-friendly and intuitive that allows users to interact with the application easily. The GUI should include options to input data, choose JAB Code symbol layouts, specify output file names, and enable/disable encryption.

   b) **Performance:** The application should process input data, apply encryption, and generate JAB Codes efficiently and without significant delays.

3. **Constraints:**

   a) **Dependency on JAB Code writer executable:** The application relies on the JAB Code writer executable or a compatible library for generating JAB Code images. This dependency must be considered when deploying the application on different platforms or environments.

   b) **Encryption key length:** The application requires encryption keys to have a length that is 128/192/256 bits for AES encryption.

4. **Others:**
    a) **Output File:** Users should be able to specify the name of the output file. The application should save the generated JAB Code image as a PNG file with the specified name and location, so user can identify the generated barcode.
    b) **Result Display:** The application should display the result of the encoding process (success or failure) in the GUI, along with any relevant information, such as error messages or additional details.

By addressing these requirements, it will develop a comprehensive and functional JAB Code application that meets the needs of users who want to encode data into JAB Code symbols with optional encryption.

3.2.10 GUI Development Process

The GUI (Graphical User Interface) development process for JAB Code application involves several steps, including design, implementation, Create Tkinter window, Pack the widget, and run the application. Here's an overview of the process:

1) **Design the GUI layout:**
    a) Plan the placement of interface elements, including input fields, buttons, labels, and menus, to ensure a user-friendly and intuitive layout.
    b) Sketch or create wireframes to visualize the layout and get feedback from potential users.
2) **Initialize Tkinter window and set properties:**
    a) Create the main application window using tk.Tk().
    b) Set the window properties, such as size and title, using the geometry() method and other window configuration options.
3) **Create and conFig. input Widgets:**
    a) **Symbol layout selection:** Create an OptionMenu widget with options for different symbol layouts, and conFig. it using a StringVar to store the selected layout.
    b) **Input file selection:** Create a Checkbutton widget to enable/disable file selection, and a Button widget to open a file dialog for choosing an input file.

c) **Output file name entry:** Create an Entry widget for the user to specify the output file name.

d) **Data input field:** Create a Text widget for the user to input data to be encoded into the JAB Code.

e) **Encryption options:** Create a Checkbutton widget to enable/disable encryption, and an Entry widget for the user to provide an encryption key.

4) **Implement Event Handler Functions:**

a) **File Chooser:** Define a function, choose_input_file(), that opens a file dialog and updates the file selection button's label with the chosen file's path.

b) **Button Click Handler:** Define a function, on_button_clicked(), that processes user input, validates settings, generates the JAB Code, and displays the result in the output text field.

5) **Pack and Position Widgets in the Main Window:**

a) Use the pack() method (or alternative geometry managers like grid() or place()) to position and organize the interface elements in the main application window.

b) Adjust widget properties as needed to control appearance, size, and position.

6) **Run the Application and Handle user Events:**

a) Start the main event loop of the application using window.mainloop().

b) Test the application to ensure that interface elements work as expected, user inputs are properly validated and processed, and the application generates the correct output based on user inputs and settings.

3.2.11 Testing of Application

Testing of the application require verifying its functionality, user interface, and error handling. Following aspects have been consider when testing the application:

a) **Interface Elements:** Check that all interface elements, such as buttons, text boxes, labels, and menus, appear correctly and are responsive to user interactions. This includes verifying that the widgets are correctly positioned, sized, and labelled.

b) **Input Validation:** Test the application's ability to handle various inputs and settings. This includes providing invalid encryption keys, empty input fields, and incorrect file formats. The application should display appropriate error messages or notifications in these cases.

c) **Functionality:** Verify that the application correctly generates JAB Code images based on user inputs and settings, such as the chosen symbol layout, input data, and encryption options. Test various combinations of settings and input data to ensure that the application behaves as expected.

d) **File Handling:** Test the application's ability to read input data from files and generate encrypted output files when the corresponding options are enabled. Ensure that the application can correctly handle various file formats, sizes, and encodings.

e) **Encryption/Decryption:** Test the application's encryption functionality by providing different encryption keys and checking that the generated JAB Code images are encrypted and decrypted correctly. Additionally, test the application's ability to handle cases where the encryption key is not provided or is of an incorrect length.

f) **Cross-Platform Compatibility:** Test the application on different platforms (e.g., macOS, Linux) and screen resolutions to ensure compatibility and consistent appearance. This includes verifying that the application's interface elements are correctly displayed and positioned, and that platform-specific commands, such as opening the generated JAB Code image, work as expected.

By thoroughly testing the JAB Code application, it can identify and address any issues or bugs, ensuring that the application provides a smooth and reliable user experience.

3.2.12 Developed Application

The final JAB Code application will be an enhanced and user-friendly tool for generating JAB Codes with the following features:

1. **Intuitive User Interface:** The application will have a well-organized and easy-to-use interface.
2. **Flexible Symbol Layouts:** Users can choose from different symbol layouts, such as U Shape, Vertical, and Horizontal, as well as any additional layouts implemented during refinement.
3. **File and Text Input:** The application will accept both text input and file input for generating JAB Codes, providing flexibility for users to encode their desired data.

4. **Encryption Support:** Users can opt to encrypt their input data using AES encryption with a user-defined key, ensuring the confidentiality of their encoded information.
5. **Progress Indicator:** A progress indicator or status message will be implemented to keep users informed about the progress of the JAB Code generation process.
6. **Cross-Platform Compatibility:** The application will be tested and optimized for different platforms (Ubuntu, macOS, Linux) and screen resolutions, ensuring a consistent user experience.
7. **Optimized Performance:** The final application will have improved performance and efficiency, thanks to code optimizations, asynchronous programming, and better use of libraries.
8. **Quality Assurance:** The application will undergo thorough testing, including unit, integration, system, and acceptance testing, to ensure its reliability, stability, and overall quality.

3.2.13 Removing Background from Plant Images

To remove the background form images of medicinal plants online tool https://www.remove.bg/ used. Fig. 3.16 shows background removed images example.

Fig. 3.16: Background Removed Medicinal Plant Images.

3.2.14 Integration of Plant Images with Generated JAB Code

To integrate the medicinal plant images with generated JAB Code, online image editor tool is user https://www.online-image-editor.com/ used, by using overlay image function new image of plant is integrated. Fig. 3.17 shows Integration of the medicinal plant image example.

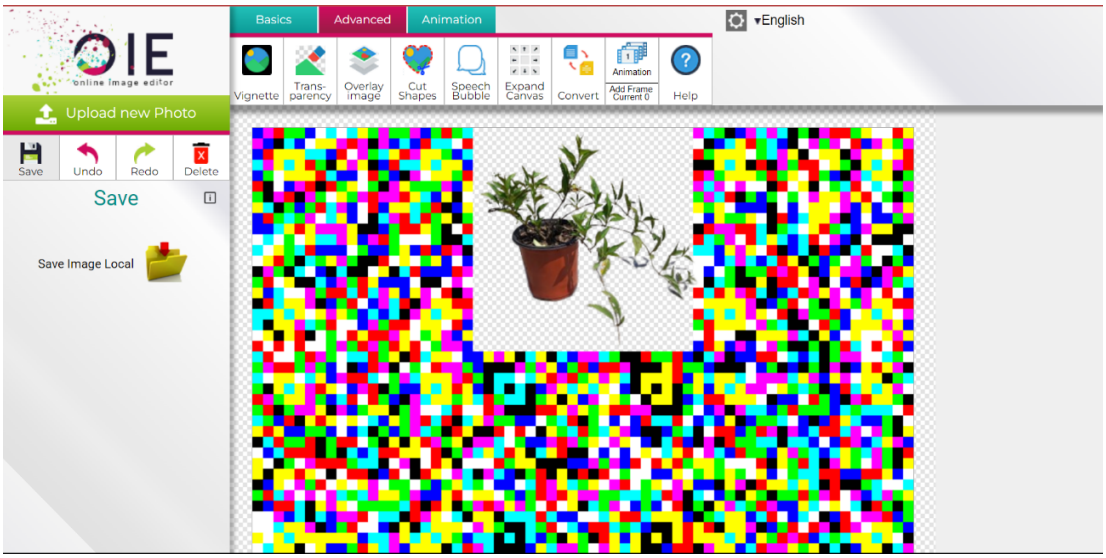

Fig. 3.17: Integration of the Medicinal Plant Image in U Shape JAB Code.

3.3 SecuJAB Generator: Application User Interaction and Input Processing

1. **Start:** This is the entry point of the application.
2. **ConFig. GUI:** Here the application sets up the Graphical User Interface (GUI) with all the necessary elements such as buttons, checkboxes, text fields, etc.
3. **Set Default Settings:** This sets the default settings for the application, such as the initial values for the text fields and default checkbox selections.
4. **Wait for Generate Button Clicked:** The application now waits for the user to press the "Generate" button to proceed.
5. **Check: Encrypt input:** This is a decision node. If the "Encrypt input" checkbox is checked, the program follows the 'Yes' path, otherwise it follows the 'No' path.
   - **Yes Path:**
     1. **Read and Validate Encryption Key:** The encryption key entered by the user is read and its validity is checked (i.e., it should be 128/192/256 bits).
     2. **Encrypt Input Text:** If the key is valid, the input text entered by the user is encrypted using the provided key.
     3. **Encrypt Input File if Selected:** If the user has chosen a file, its contents are encrypted with the same key.

- **No Path:**

    1. **Read Input Text:** If the "Encrypt input" option is not selected, the input text is read directly without encryption.
    2. **Read Input File if Selected:** If a file is chosen, its content is read without being encrypted.

6. **Build Arguments for Subprocess:** Depending on the user's choices, different arguments are prepared for a subprocess that is going to be executed.
7. **Check: Is Option Selected:** This checks the chosen option for the format of the output image. Depending on the option ("U Shape", "Horizontal", "Vertical", or "None"), it follows a different path.
    - **U Shape Path:** Set configuration specific to U Shape.
    - **Horizontal Path:** Set configuration specific to Horizontal orientation.
    - **Vertical Path:** Set configuration specific to Vertical orientation.
    - **None Path:** No extra configuration is added.
8. **Execute Subprocess:** Now, the prepared subprocess is executed with the provided arguments.
9. **Display stdout in Textview:** The output of the subprocess (stdout) is displayed in the textview in the GUI.
10. **Update Result Label to Success or Fail based on Output:** Depending on whether the subprocess was executed successfully or not, a "Success" or "Fail" message is shown in the result label.
11. **Check: Is Output a .png File:** This checks if the output file is a .png file.
    - **Yes Path:** If it is a .png file, it tries to open the file with the default image viewer on the system.
    - **No Path:** If the output file is not a .png file, it directly moves to the end of the program.
12. **End:** This is the termination point of the application.

The Flow chart of SecuJAB Generator application are shown in Fig. 3.18.

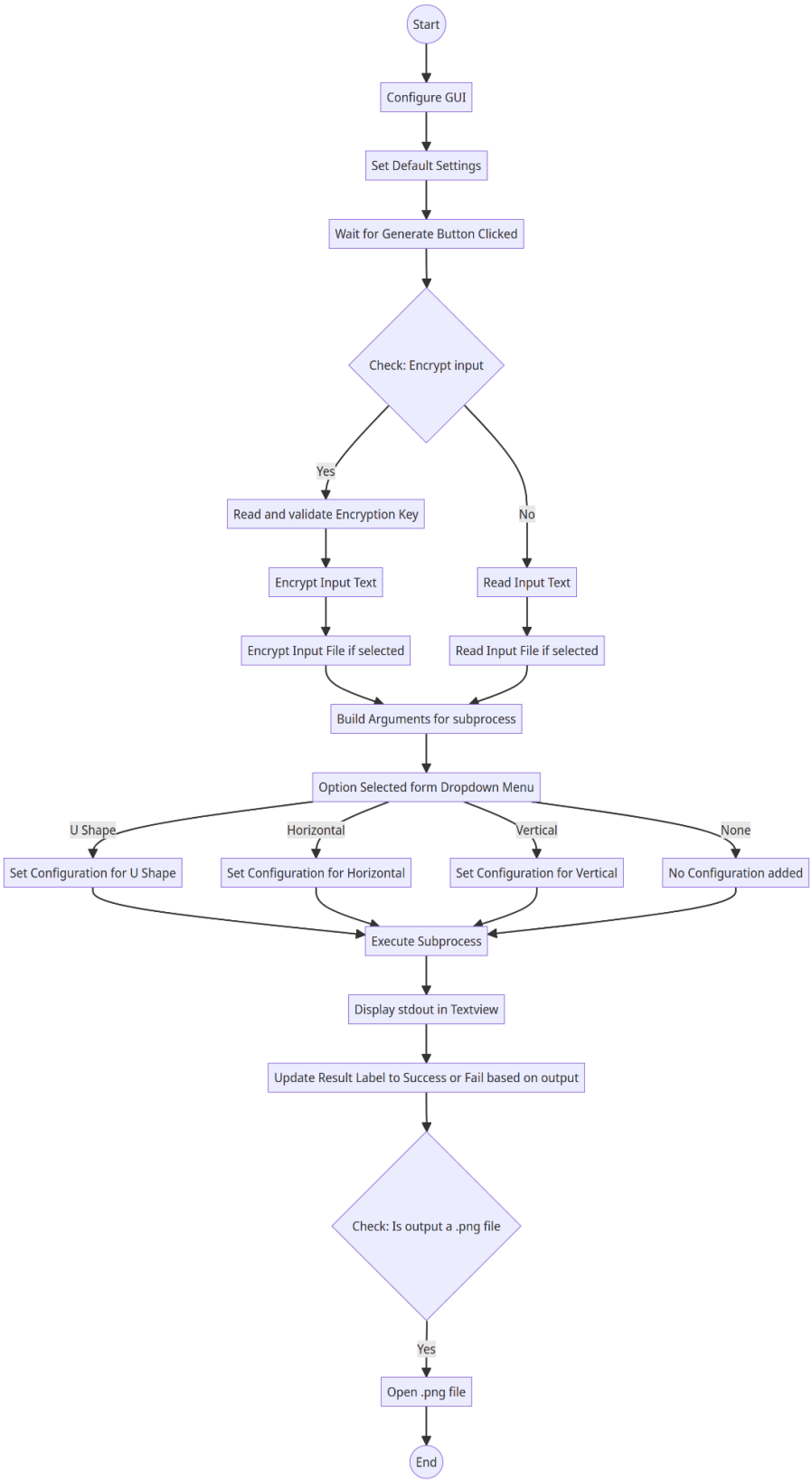

Fig. 3.18: Flow Chart for SecuJAB Generator Application

3.4 SecuJAB Reader: Application User Interaction and Input Processing

1. **Start**: This is the initial point of the process. At this stage, your application is launched and ready to operate.
2. **Select File**: The user is prompted to choose a file. This file is the data that will potentially be decrypted.
3. **Is data encrypted?**: This decision node checks if the data from the selected file is encrypted. The application determines this by attempting to interpret the data as hexadecimal a common format for encrypted data. If it can be read as hexadecimal, the data is assumed to be encrypted. Otherwise, it is assumed to be plaintext.
4. **Input Decryption Key** (Yes branch from the "Is data encrypted?" node): If the data is determined to be encrypted, the user is prompted to enter a decryption key. This key is needed to decrypt the data.
5. **Is decryption key length valid?**: This decision node checks the validity of the input decryption key. In this case a key is considered valid if its length is be 128/192/256 bits, as per the requirements of the AES encryption used in this application.
6. **Decrypt** (Yes branch from the "Is decryption key length valid?" node): If the key is valid, the application proceeds to decrypt the encrypted data.
7. **Display Decrypted Data**: The decrypted data is displayed to the user in the text widget.
8. **Display 'Invalid decryption key length' error** (No branch from the "Is decryption key length valid?" node): If the key is determined to be invalid an error message is displayed to the user, indicating that the key length is not valid.
9. **Display Data** (No branch from the "Is data encrypted?" node): If the data is not encrypted, it is directly displayed to the user in the text widget.
10. **End**: This is the final point of the process. Whether the data was encrypted or not, and whether decryption was successful or not, all paths lead to this node, marking the end of the current operation.

Flow chart of SecuJAB Reader application is shown in Fig. 3.19

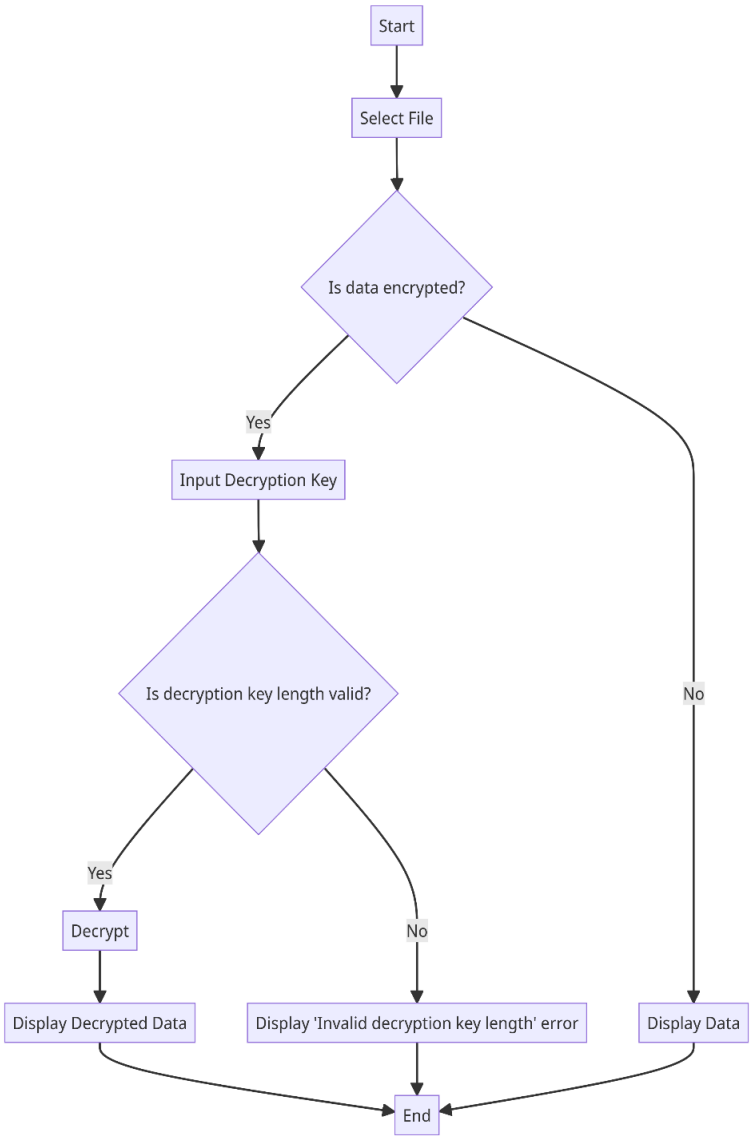

Fig. 3.19: Flow Chart for SecuJAB Reader Application

3.5 JAB Code Project Structure

JAB Code project which is available on https://github.com/jabcode/jabcode structure is shown below.

```
├── docs # Documentation
└── src # Source code
    ├── jabcode # JAB Code core library
    ├── jabcodeReader # JAB Code reader
    └── jabcodeWriter # JAB Code writer
```

3.6 Algorithm of JAB Code

1. Data preprocessing:

a) Convert input data to a byte stream.

b) Apply data compression (optional, but recommended) using algorithms like DEFLATE, LZ77, or others.

c) Encode the data using a suitable error correction code, such as Reed-Solomon or LDPC codes, to enhance the barcode's robustness.

2. Determine JAB Code parameters:

a) Select the number of colors (4 or 8) to be used in the JAB Code.

b) Determine the size (number of rows and columns) of the JAB Code matrix based on the number of input data bytes, error correction level, and the desired color count.

3. Construct the JAB Code matrix:

a) Create a matrix of size (rows x columns) and initialize it with the "Unset" state.

b) Add finder patterns, alignment patterns, and timing patterns to the matrix.

c) Encode the metadata (color count, error correction level, etc.) into the matrix using specific patterns and encoding rules.

4. Data placement:

a) Place the encoded data, including error correction bytes, into the matrix using a predefined data placement algorithm that ensures a balanced color distribution.

5. Convert the matrix to an image:

a) Map each cell value in the matrix to the corresponding color using a predefined color palette.

b) Create an image of the matrix, setting the size of each cell (module) to a fixed size, typically 1 pixel for digital representations or a small square for printed versions.

6. Postprocessing (optional):

a) Apply additional image processing techniques, such as error diffusion or color quantization, to enhance the visual appearance of the JAB Code or reduce the color count for printing.

3.7 Integration of Encryption and Decryption Functions within the Applications.

1. Encryption Process:

- Obtain the input data or file from the user interface.
- Check if the encryption check button is selected and a valid encryption key length is provided.
- If encryption is enabled:
  - Retrieve the encryption key from the input field.
  - Encrypt the input data or file using the AES encryption algorithm with Cipher Block Chaining (CBC) mode.
  - If a file is selected, read its content and encrypt it.
  - Save the encrypted data or file with an appropriate file extension (e.g., ".enc or .txt") for identification.
- If encryption is not enabled, proceed without encryption.

2. Decryption Process:

- Obtain the encrypted data or file from the user interface.
- Check if the decryption check button is selected and a valid encryption key length is provided.
- If decryption is enabled:
  - Retrieve the encryption key from the input field.
  - Decrypt the encrypted data or file using the AES decryption algorithm with Cipher Block Chaining (CBC) mode.

        - If a file was decrypted, save the decrypted content to a new file or overwrite the original file.
    - If decryption is not enabled or the provided key is invalid, display an error message.

3. Integration with JAB Code Application:

- Retrieve the selected JAB Code type from the user interface.
- ConFig. the JAB Code generation parameters based on the selected type (e.g., symbol number, symbol position, symbol version).
- Construct the command-line arguments for the external program "writer" using the JAB Code parameters and the input/output data.
- Execute the "writer" program with the constructed arguments using the subprocess.run() function.
- Capture the program output, which may include success messages or error information.
- Display the program output in the user interface for the user to view.

4. User Interface and Interaction:

- Develop a graphical user interface using Tkinter to provide a user-friendly interaction for the encryption and decryption processes.
- Design input elements such as buttons, check boxes, entry fields, and option menus for user input and configuration.
- Implement event handlers to respond to user actions, such as button clicks and check box selections.
- Display the program output and result in appropriate UI elements (e.g., text view, label).

5. Error Handling and User Feedback:

- Implement error handling mechanisms to handle potential exceptions during the encryption, decryption, and program execution processes.

- Display appropriate error messages or notifications to the user in case of failures or invalid input.
- Provide informative feedback on the encryption and decryption results, such as success/failure messages or status labels.

6. Thorough Testing and Evaluation:

- Conduct extensive testing of the application's functionalities, including encryption, decryption, JAB Code generation, and user interface interactions.
- Validate the correctness and reliability of the encryption and decryption processes using various test cases, including different input data, encryption keys, and file types.
- Evaluate the performance of the application in terms of speed, resource usage, and user experience.
- Document the testing and evaluation results to support the validity and effectiveness of the implemented encryption and decryption methodology

Results and Discussion

Chapter 4 RESULTS AND DISCUSSION

This chapter presents the results, implementation and evaluation of the proposed application. The application used for proposed work is developed using Python. The Multicolor Barcode has been generated with different multiple options. The generated barcode can be read from computer application and mobile application, while secured barcode only can read form our application.

4.1 SecuJAB Generator Application

SecuJAB Generator Application is based on python, for running the application user need to run the python file on the terminal which is not suitable for the non-technical or normal person. For easiness of the user shell script has been created, user just need to run the file and application will launch.

- Right click on the file name "SecuJAB Code Writer Application.sh" and click on Run as a Program, it will launch the application shown in Fig. 4.1.

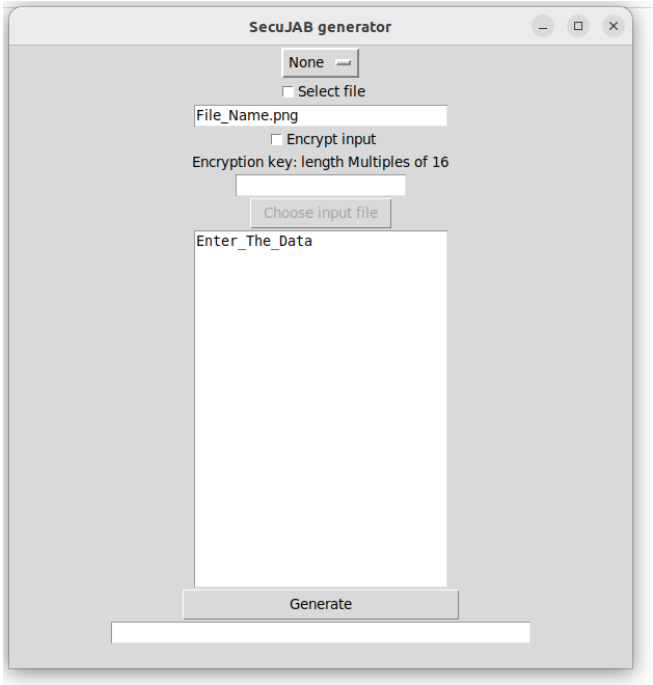

Fig. 4.1: SecuJAB Generator Application

- When the application is open user can click on Generate Button and It will create a JAB Code barcode, in this case this barcode contains "Enter_The_Data" because there is no input is given by the user only just generated a barcode.

Fig. 4.2: Generated JAB Code by SecuJAB Generator Application

- When the barcode it generated the barcode file will automatically open and shown on the widow to confirm that the barcode is generated

4.2 SecuJAB Reader Application.

SecuJAB Reader Application is based on python, for running the application user need to run the python file on the terminal which is not suitable for the non-technical or normal person. For easiness of the user shell script has been created, user just need to run the file and application will launch.

- Right click on the file name “JAB Code Reader Application.sh” and click on Run as a Program, it will launch the application shown in Fig:

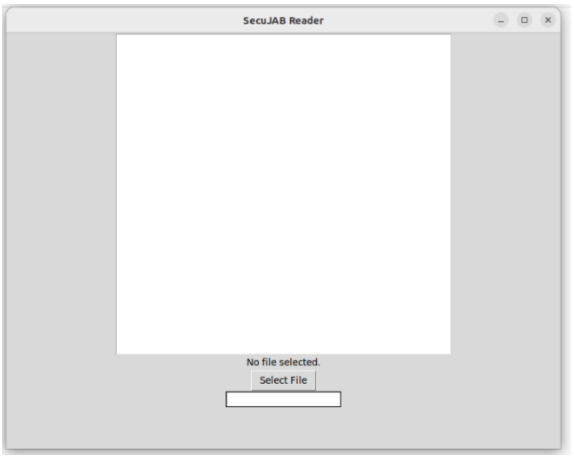

Fig. 4.3: SecuJAB Reader Application

- User can click on “Select File” to choose the barcode which can be read using this application.
- If barcode is encrypted used need to input the key to read the barcode contains.

4.3 Experimental setup of JAB Code for plant information.

This portion contains important information regarding the dataset used for information of plants which available in university for medical purpose. The experiment setup is prepared for dataset containing the Images and information of plants.

4.3.1 Structure of JAB Code used for Plant Information.

Barcode can have different structure, because for easiness of user barcode is going to contain the plant image also that's why here U shape is used for the plant information. Structure and architecture of U shape is defined in the Fig. 3.6, in Fig. 4.4 only the plant information has been stored.

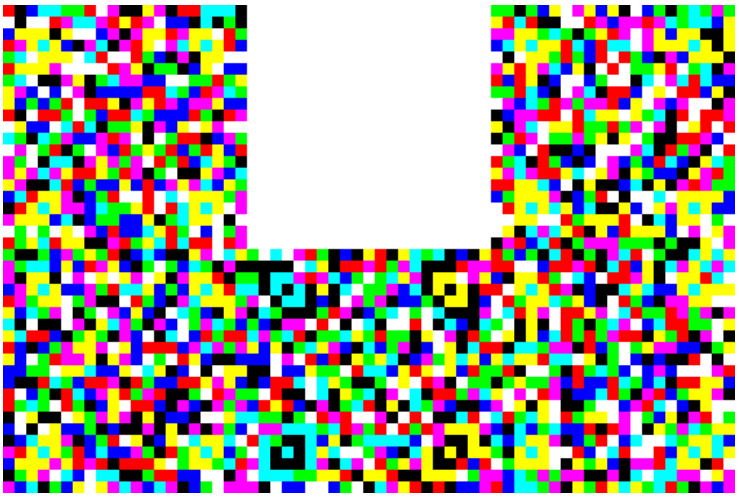

Fig. 4.4: U Shape JAB Code Containing Plant Details

As in Fig. 4.4 there is empty space where plant image can be placed, for the reference show in Fig. 4.5.

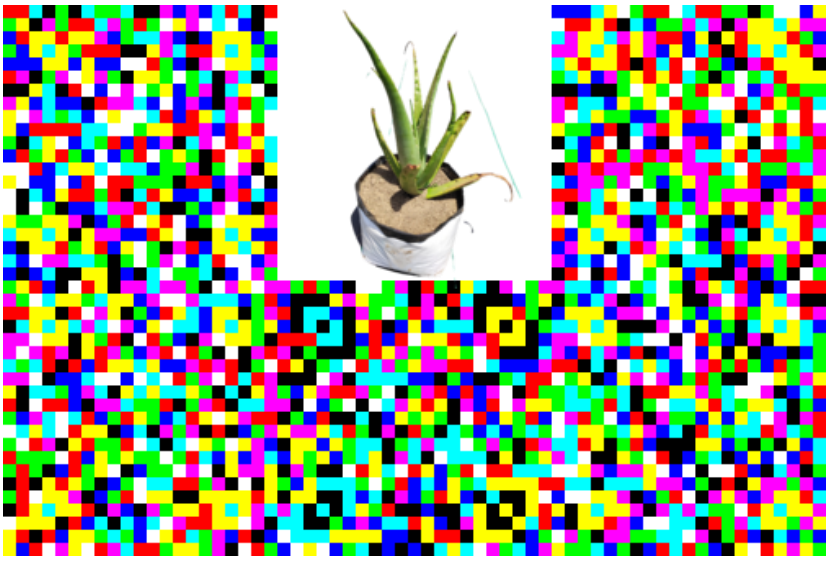

Fig. 4.5: U Shape JAB Code with Plant Image Containing Plant Details

This barcode can be scanned by our developed application and mobile application which is provide by the JAB Code developers, here are the following outputs on different sources.

a) SecuJAB Reader

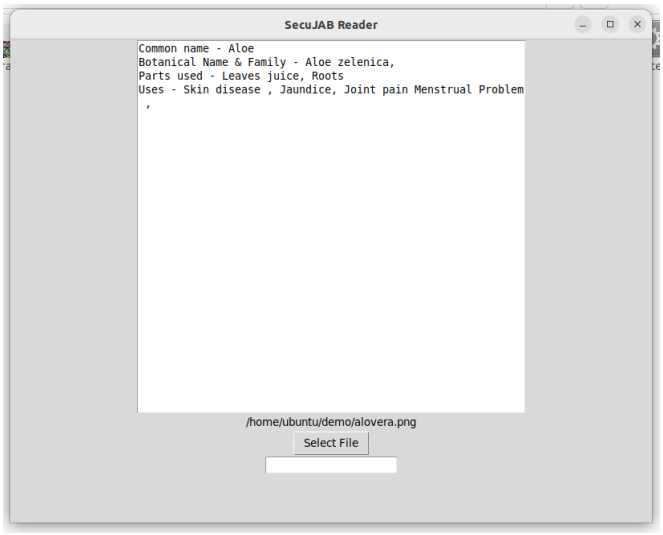

Fig. 4.6: JAB Code illustrated in Fig. 4.4 is processed using the SecuJAB Reader.

b) Mobile Application

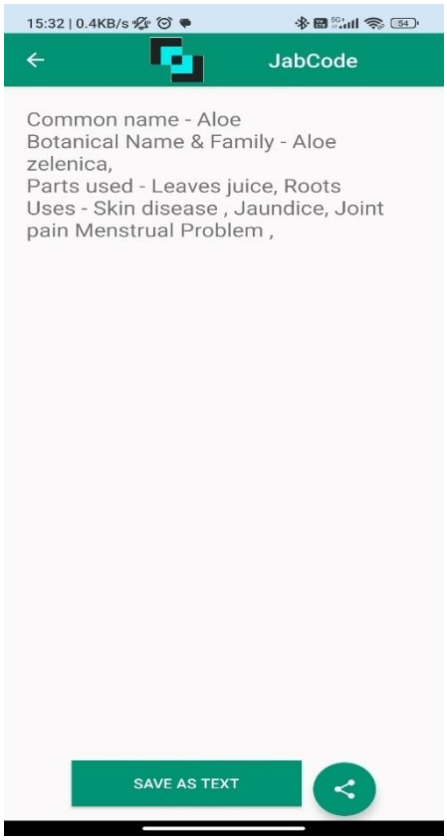

Fig. 4.7: JAB Code illustrated in Fig. 4.5 is scanned by mobile application.

4.4 Different Functionalities of SecuJAB Writer Application.

Application GUI consist with the multiple different functions, which have explained on following Fig. 4.8.

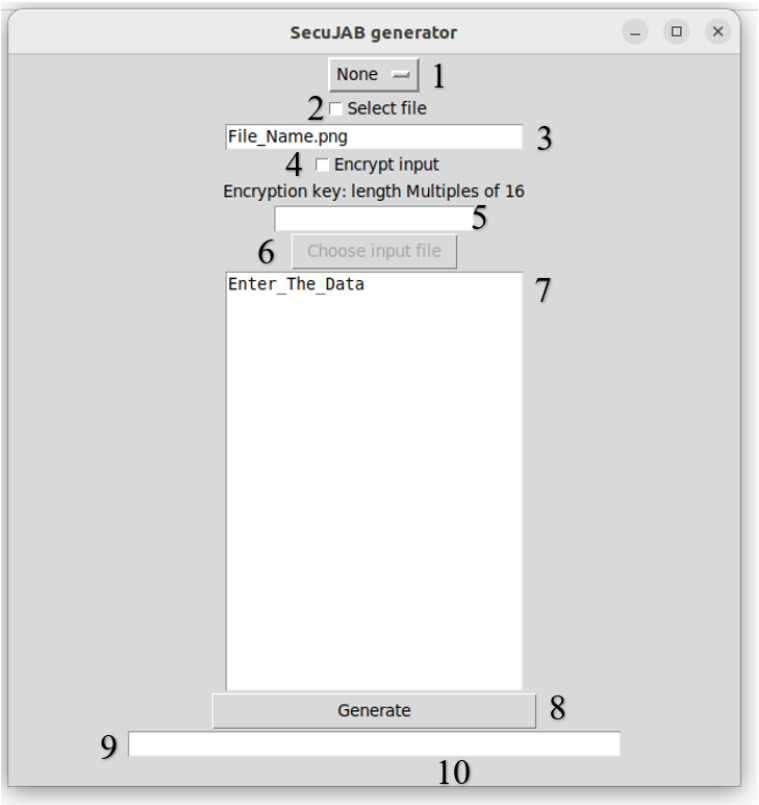

Fig. 4.8: Different function on SecuJAB application

1) Shape type – Normal Fig. 4.9, U Shape Fig. 4.18, Horizontal Fig. 4.15, vertical Fig. 4.12
2) Select File – Enable the File selection option.
3) File name – User can enter costume file name.
4) Encrypt input – Enable the encryption of the data in barcode.
5) Encryption Key – User need to put secret key which can decrypt the data, key should be 128/192/256 bits
6) Choose input file – User can select the file form this option and selected file's data going to encoded into barcode.
7) Text Area – User can type the data in the area which can converted into a barcode.

8) Generate – It will generate the barcode.

9) Error message – If any kind of error occur it will show the error message.

10) Result – It will show the Result of the operation.

4.5 Different types of JAB Code generated by SecuJAB generator Application and output by SecuJAB Reader.

1) Fig. 4.9 illustrates a standard barcode produced by the SecuJAB generator, where no options were selected and "test data without selecting any option." was inputted. Once generated, the barcode's output is captured; as displayed in Fig. 4.10 by SecuJAB reader application, and in Fig. 4.11 by a mobile application.

Fig. 4.9: Simple JAB Code

Fig. 4.10: JAB Code illustrated in Fig. 4.9 is processed using the SecuJAB Reader.

Fig. 4.11: JAB Code illustrated in Fig. 4.9 is scanned by mobile application.

2) Fig. 4.12 illustrates a different barcode produced by the SecuJAB generator, selecting vertical option and "test data with selecting vertical Shape option." was entered. When the barcode is generated, its output is produced by SecuJAB reader in Fig. 4.13, and in Fig. 4.14 by a mobile application.

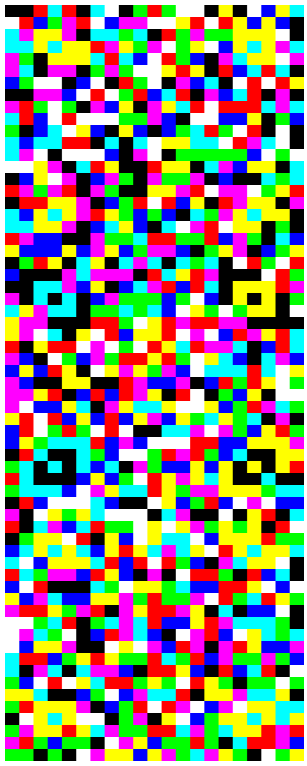

Fig. 4.12: Vertical JAB Code

Fig. 4.13: JAB Code illustrated in Fig. 4.12 is processed by SecuJAB Reader.

Fig. 4.14: JAB Code illustrated in Fig. 4.12 is scanned by mobile application.

3) Fig. 4.15 illustrates a different barcode produced by the SecuJAB Generator, selecting horizontal option and "test data with selecting horizontal Shape option." was inputted. When the barcode is generated, its output is produced by SecuJAB reader in Fig. 4.16, and in Fig. 4.17 by a mobile application.

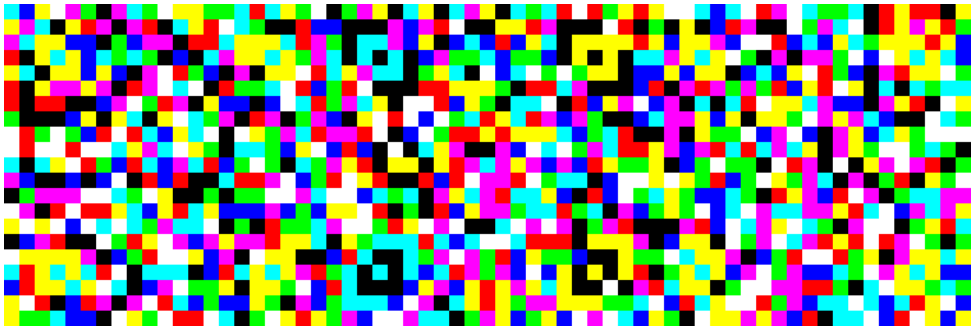

Fig. 4.15 Horizontal JAB Code

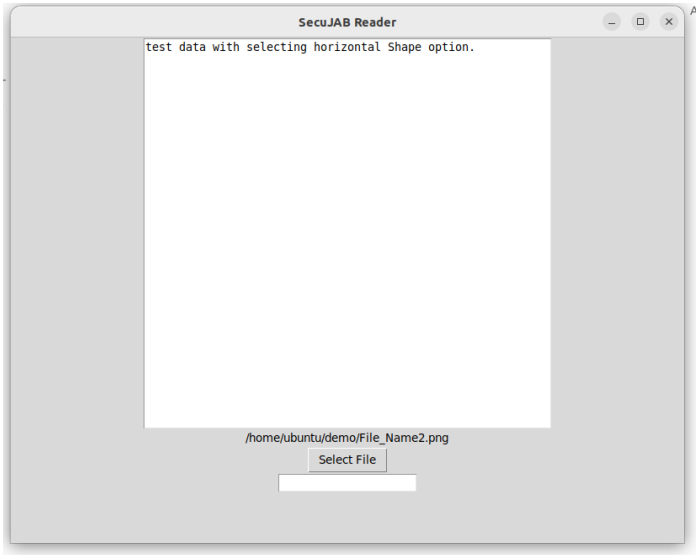

Fig. 4.16: JAB Code illustrated in Fig. 4.15 is processed by SecuJAB Reader.

Fig. 4.17: Scanning JAB Code of Fig. 4.9 using mobile application

4) Fig. 4.18 illustrates a different barcode produced by the SecuJAB generator, selecting U shape option and “test data with selecting U Shape option.” was inputted. When the barcode is generated, its output is produced by SecuJAB reader in Fig. 4.19, and in Fig. 4.20 by a mobile application.

Fig. 4.18: U shape JAB Code

Fig. 4.19: JAB Code illustrated in Fig. 4.18 is processed by SecuJAB Reader.

Fig. 4.20: JAB Code illustrated in Fig. 4.18 is scanned by mobile application.

5) Fig. 4.21 illustrates a different barcode produced by the SecuJAB generator, selecting File option and txt file was inputted containing details about fertilizer. When the barcode is generated, its output is produced by SecuJAB reader in Fig. 4.22, and in Fig. 4.23 by a mobile application.

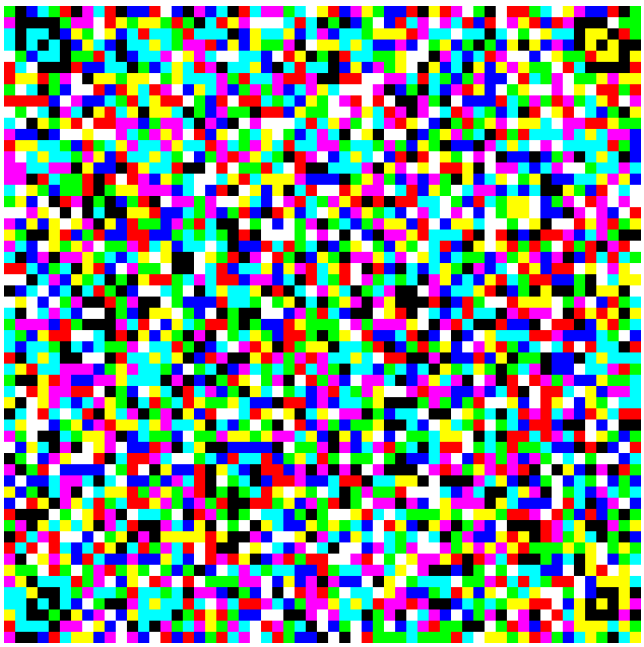

Fig. 4.21: JAB Code generated using file option

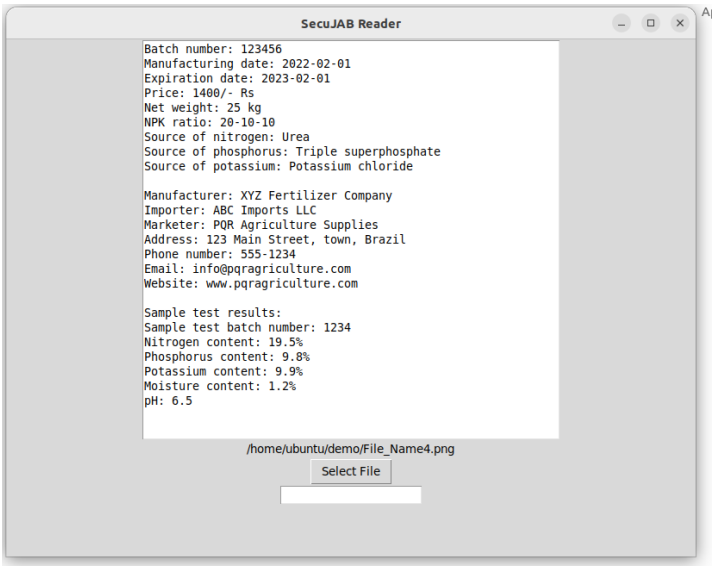

Fig. 4.22: JAB Code illustrated in Fig. 4.21 is processed by SecuJAB Reader.

Fig. 4.23: JAB Code illustrated in Fig. 4.21 is scanned by mobile application.

6) Barcode without selecting encryption option. Fig. 4.24 illustrates a different barcode produced by the SecuJAB generator, selecting encryption option and key was inputted which can decrypt the encrypted data. When the barcode is generated, its output is produced by SecuJAB reader in Fig. 4.25 which only show the encrypted data because the key hasn't been provide, in Fig. 4.26 after inputting the key the original data can be seen, and in Fig. 4.27 by a mobile application only the encrypted data can be seen.

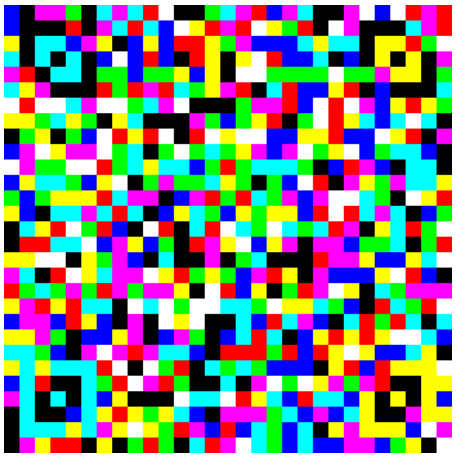

Fig. 4.24: JAB Code with encrypted option

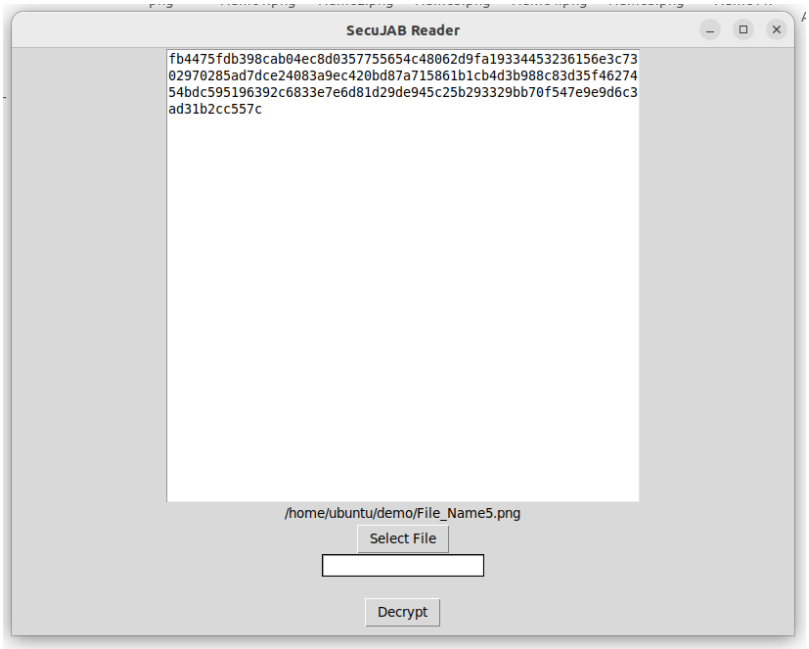

Fig. 4.25: JAB Code illustrated in Fig. 4.24 is processed by SecuJAB Reader.

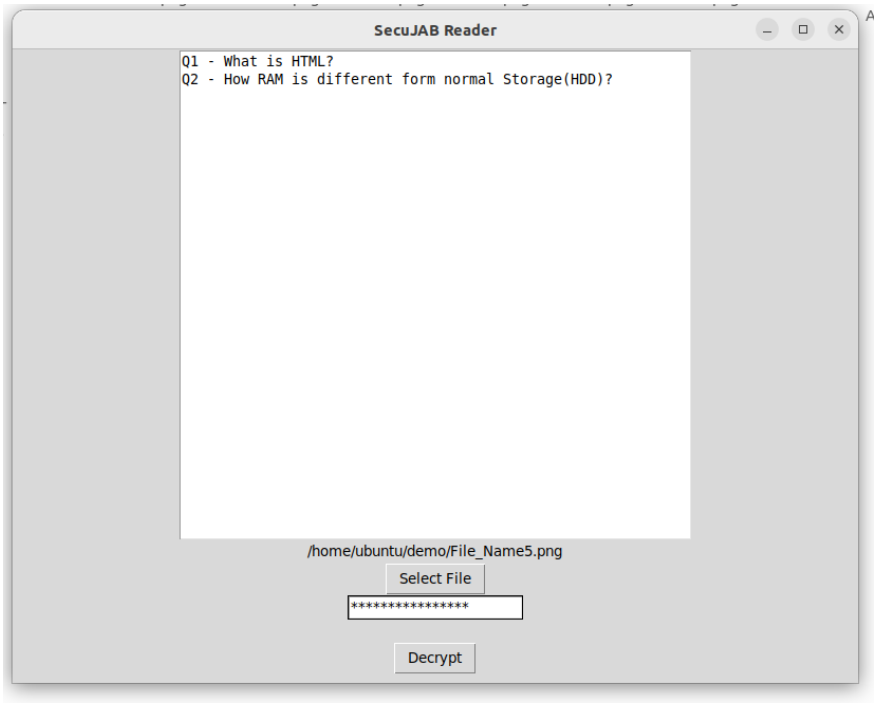

Fig. 4.26: JAB Code illustrated in Fig. 4.24 is processed by SecuJAB Reader after key.

Fig. 4.27: JAB Code illustrated in Fig. 4.24 is scanned by mobile application.

4.6. DISCUSSION

The core objective of proposed method was to develop a user-friendly application that simplifies the process of generating JAB Codes while incorporating AES encryption for enhanced data security. This objective was driven by the recognition of the potential of JAB Code technology as a high-capacity, colored matrix barcode system and the need to make it more accessible to non-technical users. Throughout the implementation, it became apparent that the successful integration of a GUI with the process of creating JAB Codes could significantly reduce the technical barriers associated with JAB Code technology. By providing an intuitive user interface and a streamlined process for generating JAB Codes, the application enhances the usability of this technology, promoting wider adoption.

The incorporation of AES encryption was another important aspect of this project. This feature directly addresses concerns related to data security, especially when encoding sensitive information into JAB Codes. By enabling the encryption of input data, the application provides an additional layer of security, thereby making JAB Code technology more appealing for various use cases that require higher levels of data protection.

In conclusion, this project has demonstrated that it is possible to create a user-friendly application that simplifies the use of JAB Code technology and enhances data security through AES encryption. This success opens up new possibilities for the application of JAB Code technology and AES encryption in a wide range of scenarios, potentially driving increased adoption and utilization of these technologies in the future.

Summary and Conclusion

Chapter 5 SUMMARY AND CONCLUSION

5.1. Summary

In this proposed method, a desktop application is developed using Python's tkinter library for GUI creation, providing users a platform to interact with JAB Code (Just Another Barcode) encoding and decoding processes. The goal of this application is to provide a simple and intuitive interface for users to interact with the underlying JAB Code technology.

The JAB Code, a 2D colored matrix barcode, is a recent development in data representation technology, offering higher data capacity and robustness against color distortion compared to traditional black and white 2D barcodes. However, usability has been a limiting factor in its widespread adoption. the application addresses this issue by providing a user-friendly interface to perform operations such as generating JAB Code from user input or files and selecting the pattern in which the JAB Code is generated.

One significant feature of this application is the incorporation of AES encryption. This feature allows the user to encrypt their input data, thereby enhancing the security of the information being encoded in the JAB Code. The application allows for the selection of the encryption option and facilitates the entry of an encryption key.

The program uses subprocess module to interact with the system's command line, allowing the use of an existing JAB Code writer program in a user-friendly manner. It also automatically opens the generated JAB Code image file using the most suitable command depending on the user's operating system.

To summarize, this application significantly simplifies the process of creating JAB Codes, while also enhancing security through AES encryption. It also brings JAB Code technology closer to non-technical users, thus promoting its widespread adoption. This desktop application is a bridge between complex data encoding/decoding technology and everyday users who may benefit from using such technologies.

5.2. Conclusion

In this research project, successfully developed a user-friendly application to enable easy use of JAB Code, a high-capacity colored matrix barcode system. The application not only demystifies the process of creating JAB Codes but also integrates AES encryption, thus enhancing the security of the encoded data.

The key accomplishments of this proposed method can be summarized as follows:

1) Developed an easy-to-use GUI application to interact with the JAB Code system, thereby making this technology accessible to non-technical users.
2) Incorporated AES encryption for enhancing data security, with a provision for users to input an encryption key of their choice.
3) Provided flexibility in terms of user inputs, allowing both text and file input options, and facilitated various JAB Code pattern choices.

The application marks a significant stride towards increasing the adoption of JAB Code technology. By incorporating a user-friendly interface and essential security features, we have demonstrated that advanced encoding technologies such as JAB Code can be made accessible and secure for everyday users. This approach could serve as a model for making other advanced technologies more user-friendly in the future.

5.3 Future Scope

The developed application has successfully demonstrated a user-friendly approach to harnessing the potential of JAB Code technology with integrated AES encryption. While it presents a significant stride in making such technology accessible and secure for everyday users, it simultaneously lays a foundation for future enhancements and applications. One immediate advancement can be integrating an in-built JAB Code writer into the application, eliminating the dependency on an external one, hence making the application more self-contained and portable. Also, while AES encryption provides robust data security, incorporating a range of encryption algorithms like RSA, Blowfish, and Twofish, among others, would offer users a more versatile array of security options to suit their specific needs.

The current application version focuses on encryption and generation of JAB Codes. Expanding this to include a decryption module would provide a holistic solution for users, allowing them not only to generate but also decode encrypted JAB Code data. Further customization options for the JAB Code, such as different shapes, sizes, and color palettes, could also be included to cater to users' aesthetic requirements. Beyond the confines of the current desktop application, a mobile version could be developed. This would open up JAB Code generation and encryption to an even wider audience, as smartphones are becoming increasingly ubiquitous. Also, integrating our application with various online and offline services, like data storage platforms, email, messaging,

and other communication applications, would allow for secure data transfers in a multitude of contexts.

Ultimately, the continued evolution and adaptation of this application to users' changing needs and emerging technologies will ensure that JAB Code and AES encryption continue to serve as efficient, secure, and user-friendly tools for data encoding and decoding.

5.3.1 Some other use case of JAB Code

1. User can add their signature to add more authenticity, example is show in Fig. 5.1.

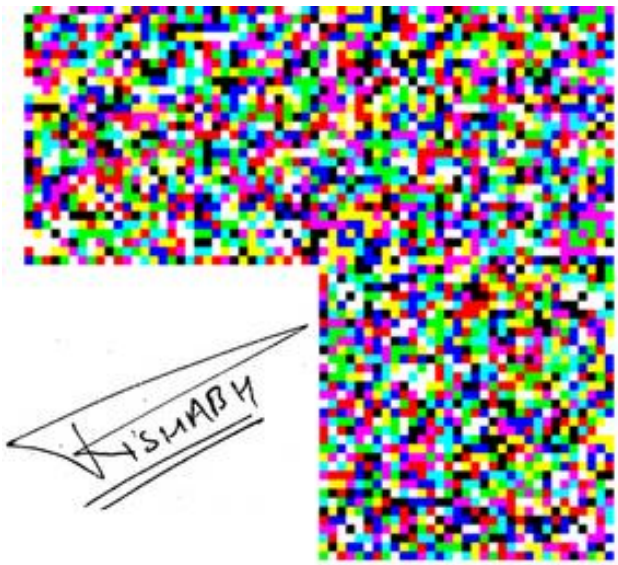

Fig. 5.1: JAB Code containing user signature in free space

2. Organization can add their own hologram for the authenticity, example is shown in Fig. 5.2.

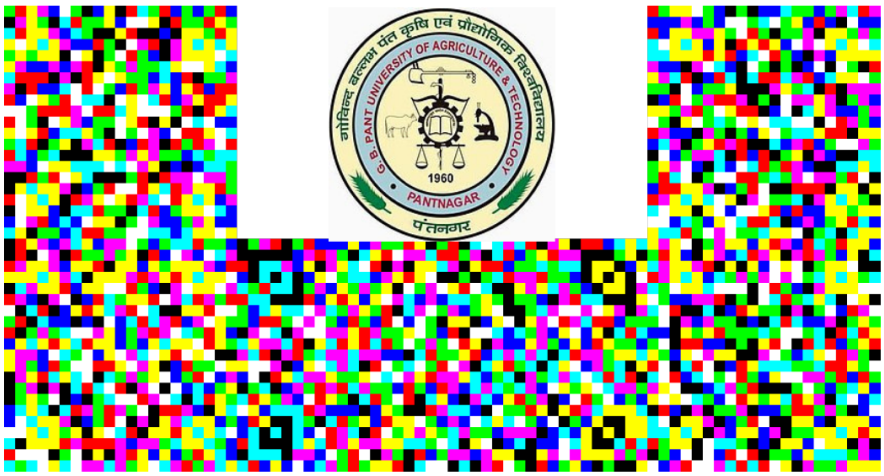

Fig. 5.2: JAB Code containing hologram

3. Certificate containing the whole certificate information, which can help in offline verification, example is shown in Fig. 5.3.

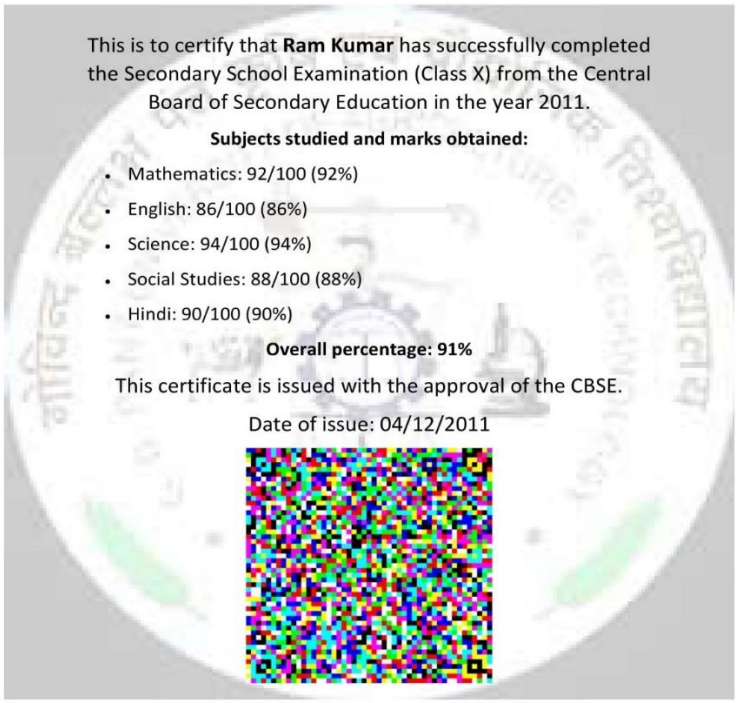

Fig. 5.3: Certificate containing the JAB Code

5.4 Refine and Improve for Future

Here are more details on refining and improving the application under the given categories:

1. **Enhance Error Handling:**
   a) Add more specific error messages for different error scenarios, such as incorrect encryption key length, empty input fields, or unsupported file formats.
   b) Use exception handling to catch and display errors in a user-friendly manner, guiding the user on how to resolve the issues.
2. **Expand Encryption Functionality:**
   a) Allow users to choose from different encryption algorithms or key lengths, providing more flexibility and security options.
   b) Implement an option to generate a random encryption key for users who may not have a specific key in mind.

3. **Improve User Experience:**
   a) Reorganize the interface layout to make it more intuitive and user-friendly.
   b) Adjust widget sizes and positions for better readability and usability.
   c) Add tooltips or help messages to guide users on how to use the application effectively.
4. **Introduce Additional Symbol Layouts:**
   a) Offer more customization options for users by adding new symbol layouts.
   b) Implement a custom layout feature, allowing users to define the number and position of symbols according to their specific requirements.
5. **Implement Progress Indicator:**
   a) Add a progress indicator or status message to inform the user about the progress of the JAB Code generation process, especially for larger inputs or files that may take longer to process.
   b) Use threading or asynchronous programming techniques to prevent the application from freezing during the generation process.
6. **Enable Saving Settings:**
   a) Implement a feature to save user preferences and settings, either in a configuration file or by using the operating system's registry.
   b) Allow users to load their saved settings to streamline the process and improve efficiency.
7. **Add More Export Options:**
   a) Provide additional export options for the generated JAB Code images, such as different file formats (e.g., JPEG, BMP, SVG) or adjustable image resolution.
   b) Implement the option to export the JAB Code as a standalone file or embedded in a PDF or other document formats.
8. **Optimize Code:**
   a) Review the code to identify areas for performance improvement, maintainability, and readability.
   b) Refactor the code, remove redundancies, and use more efficient algorithms or libraries where appropriate.

Literature Cited

LITERATURE CITED

Berchtold, W., Liu, H., Steinebach, M., Klein, D., Senger, T. and Thenee, N. 2020. JAB Code-A Versatile Polychrome 2D Barcode. *Electron. imag.*, 2020(3): 1-6.

Bulan, O., Monga, V. and Sharma, G. 2009. High capacity color barcodes using dot orientation and color separability. '*In: Media forensics and security'* at San Jose, California, United States, during. January 18-22. pp. 397-403.

Dey, S., Agarwal, S. and Nath, A. 2013. Confidential encrypted data hiding and retrieval using qr authentication system. '*In: International Conference on Communication Systems and Network Technologies (CSNT)'* at Gwalior, India, during. April 6-8. pp. 512-517.

Dworkin, M. J., Barker, E. B., Nechvatal, J. R., Foti, J., Bassham, L. E., Roback, E. and Dray Jr, J. F. 2001. *Advanced encryption standard (AES).* 2001 :1-52

Grillo, A., Lentini, A., Querini, M. and Italiano, G. F. 2010. High capacity colored two dimensional codes. '*In: Proceedings of the international multiconference on computer science and information technology'* at Wisla, Poland, during. October 18-20. pp. 709-716.

Mittra, P. and Rakesh, N. 2016. A desktop application of QR code for data security and authentication. '*In: International Conference on Inventive Computation Technologies (ICICT)'* at Coimbatore, India, during. August 26-27. pp. 1-5.

Parikh, D. and Jancke, G. 2008. Localization and segmentation of a 2D high capacity color barcode. '*In: 2008 IEEE workshop on applications of computer vision'* at Copper Mountain, CO, USA, during. January 7-9. pp. 1-6.

Shao, F., Chang, Z. and Zhang, Y. 2010. AES encryption algorithm based on the high performance computing of GPU. '*In: 2010 Second International Conference on Communication Software and Networks'* at Singapore, Singapore, during. February 26-28. pp. 588-590.

Winter, C., Berchtold, W. and Hollenbeck, J. N. 2019. Securing physical documents with digital signatures. '*In: IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)'* at Kuala Lumpur, Malaysia, during. September 27-29. pp. 1-6.

Yfantis, V., Kalagiakos, P., Kouloumperi, C. and Karampelas, P. 2012. Quick response codes in E-learning. '*In: International Conference on Education and e-Learning Innovations'* at Sousse, Tunisia, during. July 1-3. pp. 1-5.

Appendices

APPENDICES

JAB Code for Medicinal Plants

Medicinal plants which are available at the university their images taken manually. Fig. 3.2 show the example of the images. By using images and JAB code together we are interested to provide following details.

1. **Common Name**: This is the name commonly used to refer to the plant in everyday language, as opposed to its scientific or botanical name.
2. **Botanical Name & Family**: The botanical name is the formal scientific name following the binomial nomenclature system, consisting of the genus and species of the plant. The family is a higher taxonomic rank in the hierarchy, grouping together related plants.
3. **Parts Used**: This column specifies which parts of the plant are used in medicinal or other applications. This can include leaves, roots, seeds, flowers, etc.
4. **Uses**: This describes the medicinal or other benefits of the plant. It may include its use as a treatment for specific health conditions, or other benefits.

Generated JAB Codes of Medicinal Plants

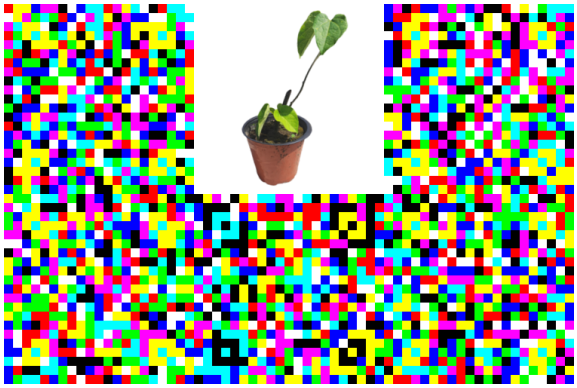

Fig. 1: Giloe

Fig. 2: Ratti

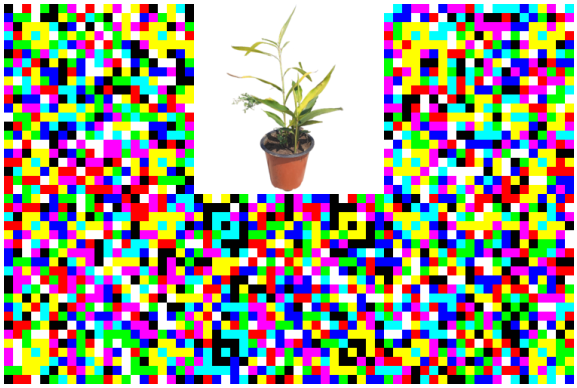

Fig. 3: Elaichi Grass

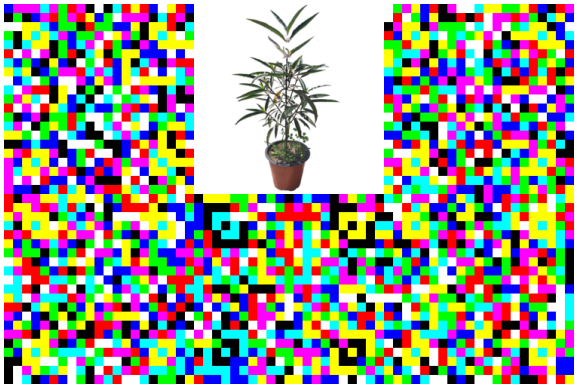

Fig. 4: Katila Bans

Fig. 5: Pilabansa

Fig. 6: Patherchur

Fig. 7: Chitrak

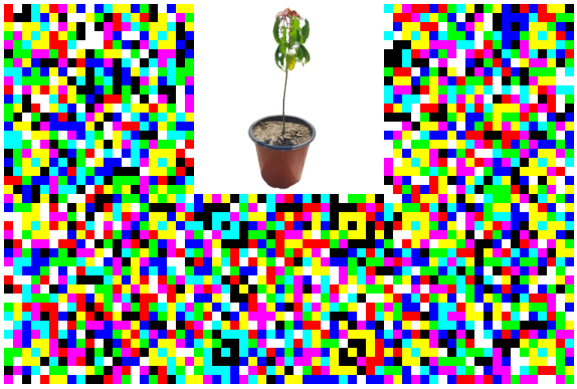

Fig. 8: Sarpgandha

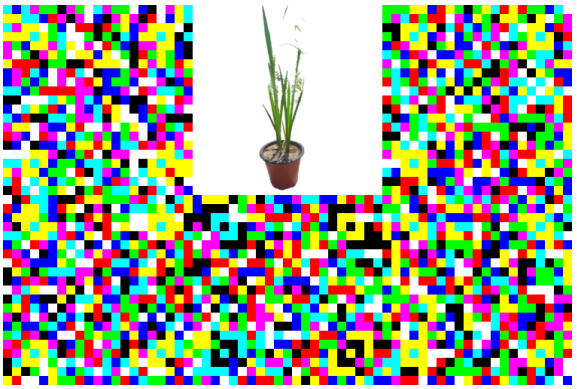

Fig. 9: Bach

Fig. 10: Achilia

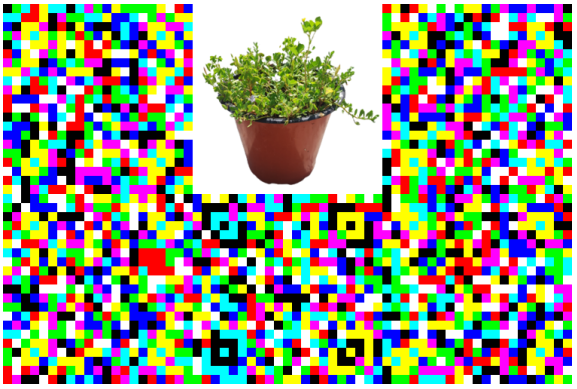

Fig. 11: Brahmi

Fig. 12: Gandh Prasarni

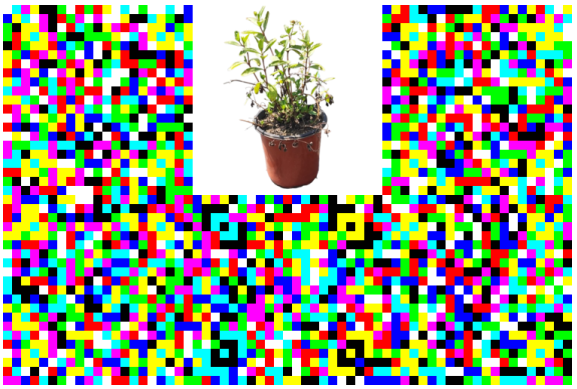

Fig. 13: Bhringaraj

Fig. 14: Ajwain Patta

Fig. 15: Manduk Parni

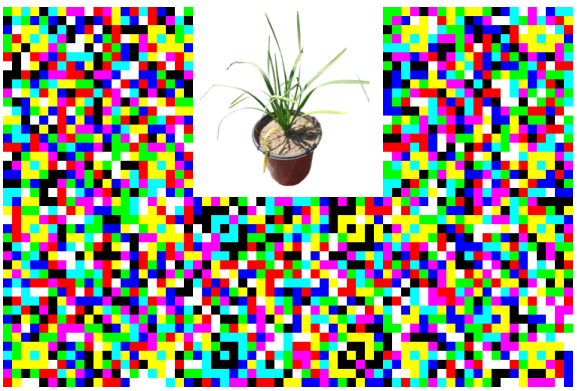

Fig. 16: Pahari Lahsun

Fig. 17: Stevia

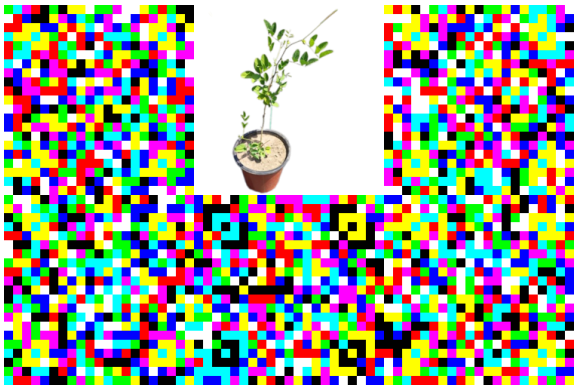

Fig. 18: Aparajita

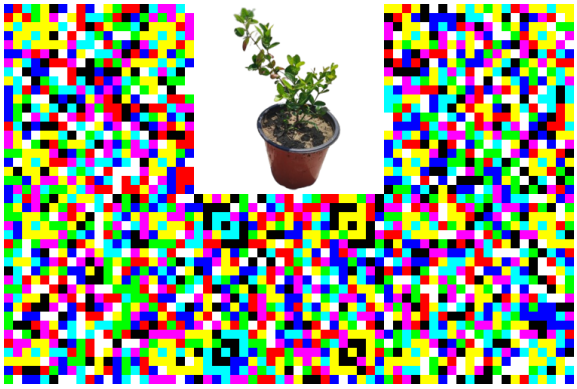

Fig. 19: Karonda

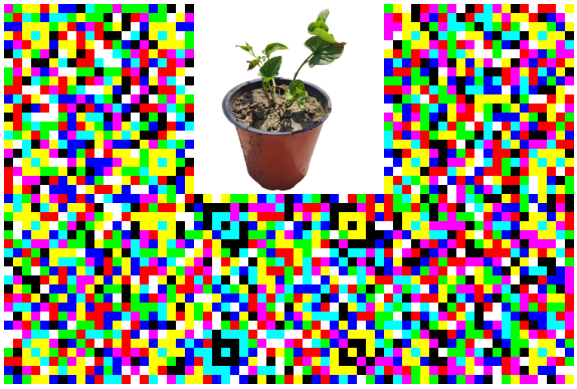

Fig. 20: Pipali

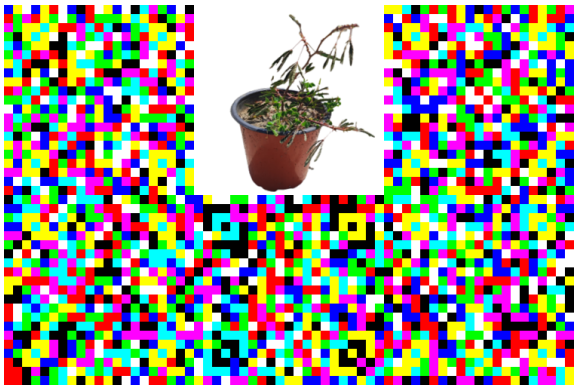

Fig. 21: Lajwanti

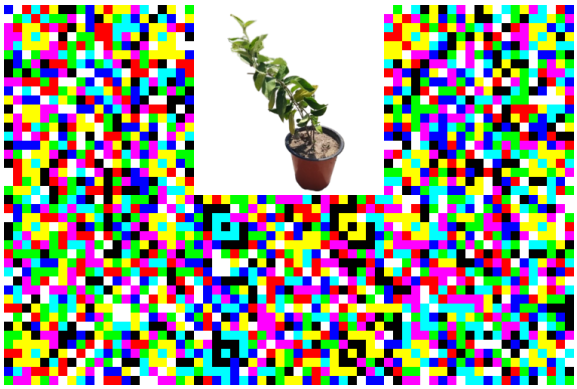

Fig. 22: Antmool

Fig. 23: Sansiveria

Fig. 24: Kalmegh

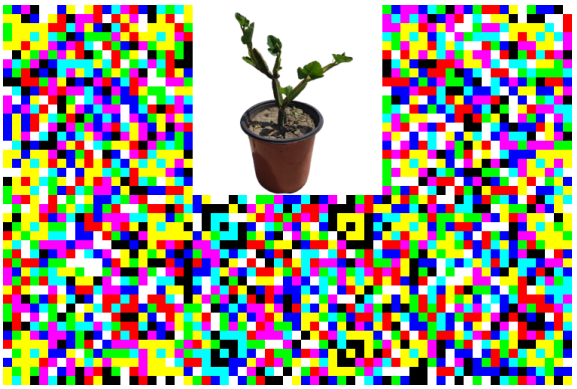

Fig. 25: Hadjod

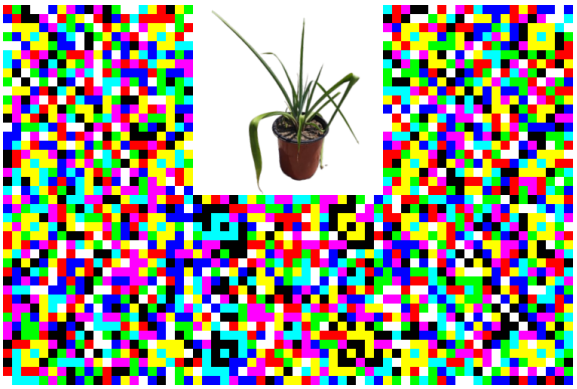

Fig. 26: Rajnigandha

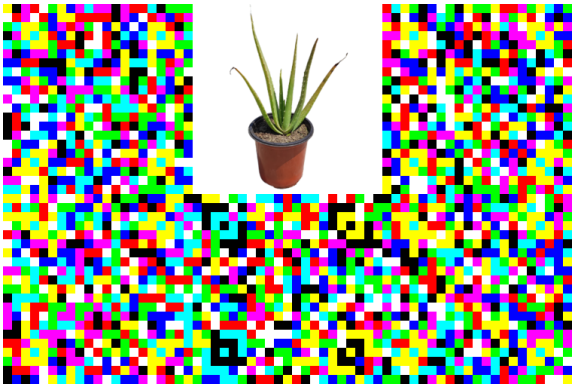

Fig. 27: Aloe

Fig. 28: Punarnava

Fig. 29: Safed Sataver

Fig. 30: Syama Tulasi

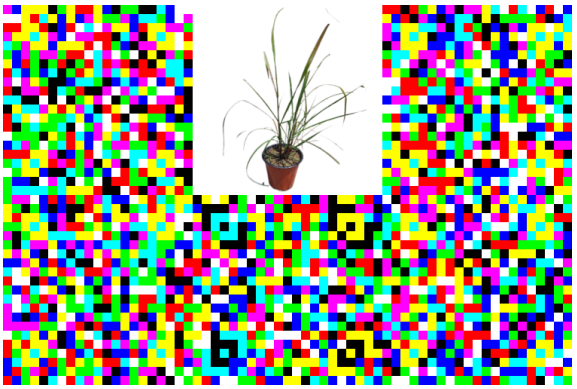

Fig. 31: Lemon Grass

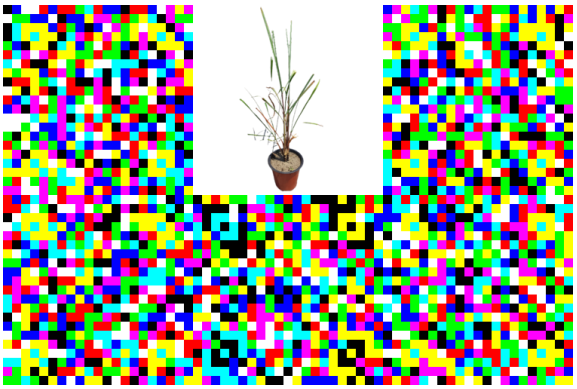

Fig. 32: Citronella

Fig. 33: Madhumalti

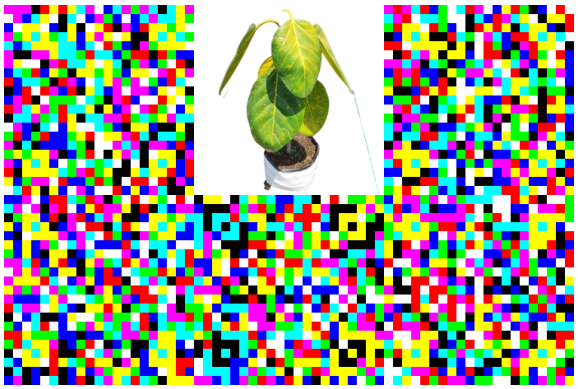

Fig. 34: Bargad

Fig. 35: Moringa Sahjan

Fig. 36: Sita Ashoka

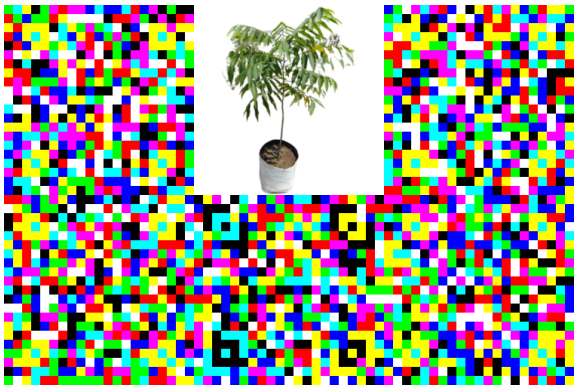

Fig. 37: Ratti

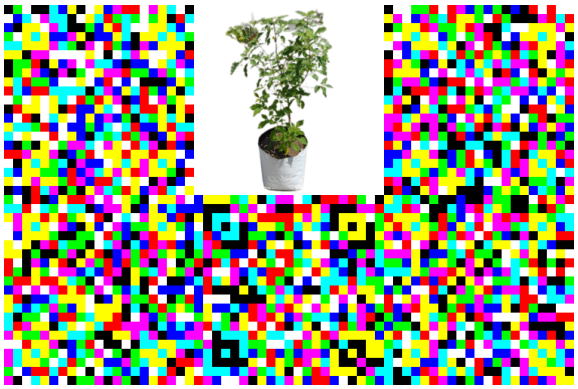

Fig. 38: Nirgundi

Fig. 39: Jamun

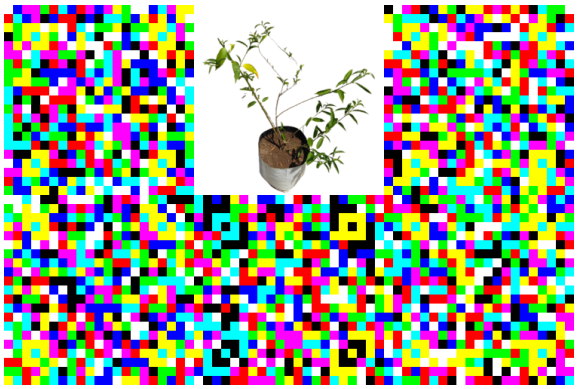

Fig. 40: Raat ki Rani

Fig. 41: Maulsari

Fig. 42: Pili Kaner

Fig. 43: Putranjeeva

Fig. 44: Sinduri

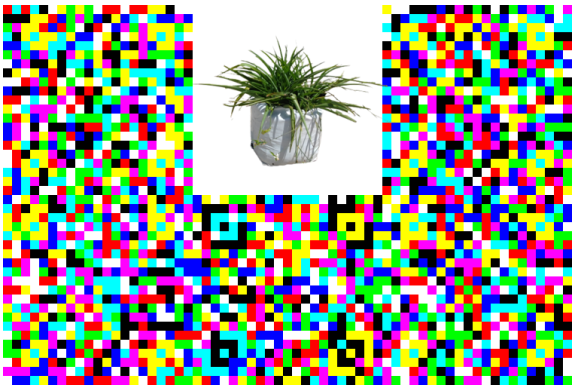

Fig. 45: Safed Musli

Fig. 46: Bakain

Fig. 47: Bahera

Fig. 48: Mehndi

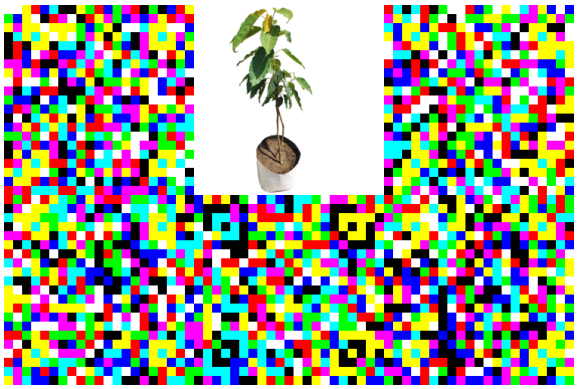

Fig. 49: Pakad

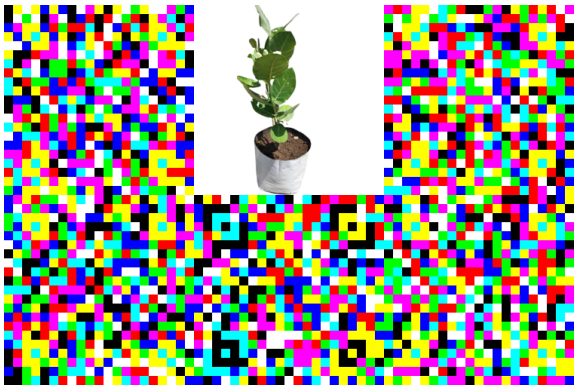

Fig. 50: Madar

Fig. 51: Arjun

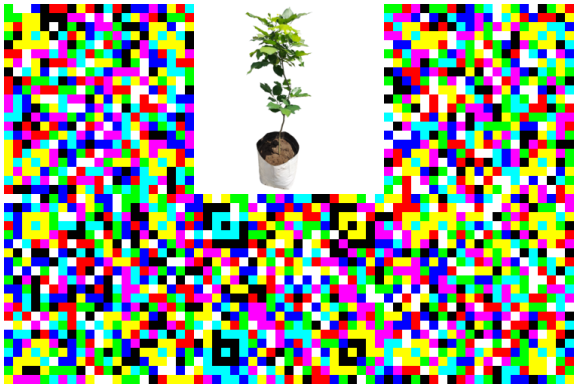

Fig. 52: Karanj

Fig. 53: Bidhara

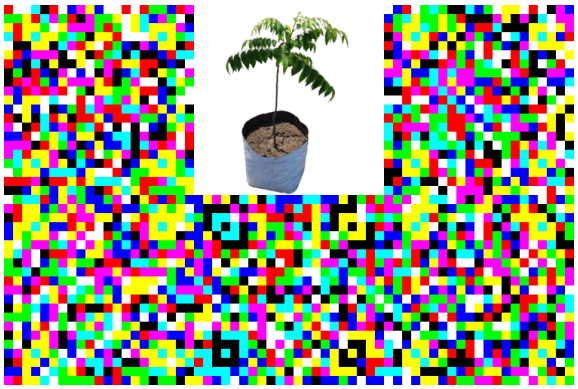

Fig. 54: Karipatta

Fig. 55: Sitaphal

Fig. 56: Aonla

Fig. 57: Camphor

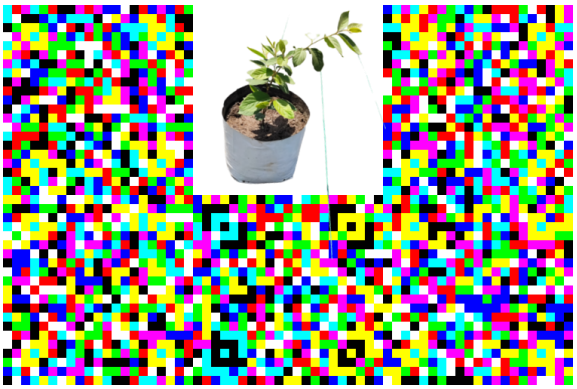

Fig. 58: Harar

Fig. 59: Nagkeser

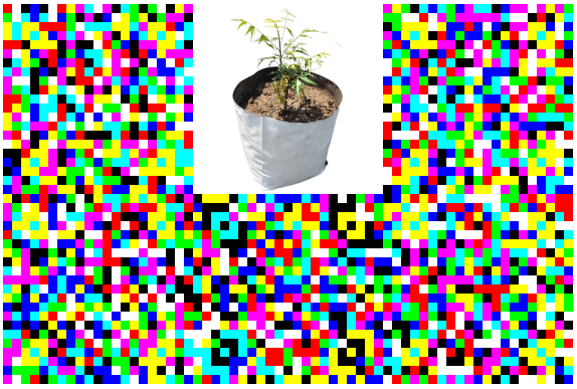

Fig. 60: Neem

Fig. 61: Dalchini

CURRICULUM VITAE

Name : Rishabh Kushawah

Phone Number : 8909426497

Mailing Address : 122, Taxi Stand Road, Civil Lines, Rampur, Uttar Pradesh, 244901

Permanent Address : 22, Taxi Stand Road, Civil Lines, Rampur, Uttar Pradesh, 244901

E-mail : kusrishabh@gmail.com

Career Objective : To work as a DevOps Engineer as a Freelancer.

Educational Qualification: (should be written in reverse chronological order)

S. No.	Examination passed	Institution	Year	Percentage/ CGPA
1.	M. Tech. (Computer Engineering)	G.B.P.U.A&T, Pantnagar	2023	8.053
2.	B. E. (Computer Science & Engineering)	CHANDIGARH UNIVERSITY, Mohali, Punjab	2019	7.0
3.	Intermediate	Saraswati Vidya Mandir Inter College, Rampur	2013	66.40%
4.	High school	Saraswati Vidya Mandir Inter College, Rampur	2011	51.00%

Specialization: Major: Computer Engineering **Minor:** NIL

Thesis Title: Creation of AES-Enabled Secure JAB-Code for Color Barcode Applications and it's Integration with Medicinal Plants Information.

Conferences/ Seminars/Workshops/Trainings Attended:

1. DevOps on AWS (Specialization) – By Amazon Web Services
2. Google IT Support (Specialization) – By Google
3. Python for Everybody (Specialization) – By University of Michigan
4. Blockchain Basics – By University at Buffalo
5. Data Science – By Internshala

Software Skill: AWS, Dev-Ops, IPFS, WordPress, Blockchain, Decentralization Tech.

1. Own Private website (All4youz.com) WordPress based website, Sub-Domains, c-panel hosting and SEO.
    [1] SEO (Search engine optimization) for better web search results.
    [2] Design Optimization (lightweight home page).
    [3] Security optimization (prevention form attackers and bots).
2. Advanced knowledge of Hardware, Operating Systems, Networks and Internet Technologies, and Hosting Local Servers.

Professional Skills: Leadership skill, Communication skill, Quick learner.

Place: Pantnagar
Date:

(Rishabh Kushawah)

Name	**:**	Rishabh Kushawah	**Id No.**	**:**	57979
Semester & Year of Admission	**:**	1st, 2021	**Degree**	**:**	M. Tech.
Major	**:**	Computer Engineering	**Department**	**:**	Computer Engineering
Thesis title	**:**	**Creation of AES-Enabled Secure JAB-Code for Color Barcode Applications and it's Integration with Medicinal Plants Information.**			
No. of pages	**:**	**72**	**Advisor**	**:**	Jalaj Sharma

ABSTRACT

This thesis presents the development of a desktop application designed to simplify interactions with JAB Code (Just Another Barcode) technology, a 2D colored matrix barcode system with superior data capacity and robustness against color distortion. The application incorporates a Graphical User Interface (GUI) developed using Python's tkinter library, rendering the complex process of encoding and decoding JAB Codes accessible to non-technical users. Notably, the application also integrates Advanced Encryption Standard (AES) encryption, empowering users to securely encode their data within the JAB Code, thereby enhancing the security of the encoded information. The program leverages the subprocess module to interact seamlessly with the command line of the system, thus simplifying the usage of existing JAB Code writer programs. The versatility of the application is demonstrated through its capacity to accept both text and file inputs and facilitate varied JAB Code pattern choices. Overall, this research project signifies a significant step towards promoting the widespread adoption of JAB Code technology by marrying complex encoding technologies with user-friendly interfaces and robust security measures.

(Jalaj Sharma)
Advisor

(Rishabh Kushawah)
Author

सेमेस्टर और प्रवेश वर्ष	:	प्रथम, 2021-2022	**उपाधि**	:	एम.टेक.
मुख्य विषय	:	कंप्यूटर इंजीनियरिंग	**विभाग**	:	कंप्यूटर इंजीनियरिंग
शोध शीर्षक	:	**"कलर बारकोड अनुप्रयोगों के लिए एईएस-सक्षम सुरक्षित जेएबी कोड का निर्माण और औषधीय पौधों की जानकारी के साथ इसका एकीकरण "**			
पृष्ठ संख्या	:	**72**	**सलाहकार**	:	**जलज शर्मा**

सारांश

यह थीसिस जे. ए. बी. कोड (जस्ट अनोत्तर बारकोड) तकनीक के साथ इंटरैक्शन को सरल बनाने के लिए डिज़ाइन किए गए एक डेस्कटॉप एप्लिकेशन के विकास को प्रस्तुत करती है, जो बेहतर डेटा क्षमता और रंग विरूपण के खिलाफ मजबूती के साथ एक 2डी रंगीन मैट्रिक्स बारकोड प्रणाली है। एप्लिकेशन में पायथन की टिंकर लाइब्रेरी का उपयोग करके विकसित एक ग्राफिकल यूजर इंटरफेस (जीयूआई) शामिल है, जो गैर-तकनीकी उपयोगकर्ताओं के लिए जेएबी कोड को एन्कोडिंग और डिकोड करने की जटिल प्रक्रिया को सुलभ बनाता है। विशेष रूप से, एप्लिकेशन उन्नत एन्क्रिप्शन स्टैंडर्ड (एईएस) एन्क्रिप्शन को भी एकीकृत करता है, जो उपयोगकर्ताओं को जेएबी कोड के भीतर अपने डेटा को सुरक्षित रूप से एन्कोड करने के लिए सशक्त बनाता है, जिससे एन्कोडेड जानकारी की सुरक्षा बढ़ जाती है। प्रोग्राम सिस्टम की कमांड लाइन के साथ निर्बाध रूप से इंटरैक्ट करने के लिए सबप्रोसेस मॉड्यूल का लाभ उठाता है, इस प्रकार मौजूदा जेएबी कोड राइटर प्रोग्राम का उपयोग सरल हो जाता है। एप्लिकेशन की बहुमुखी प्रतिभा को टेक्स्ट और फ़ाइल इनपुट दोनों को स्वीकार करने और विभिन्न जेएबी कोड पैटर्न विकल्पों की सुविधा प्रदान करने की क्षमता के माध्यम से प्रदर्शित किया जाता है। कुल मिलाकर, यह शोध परियोजना जटिल एन्कोडिंग प्रौद्योगिकियों को उपयोगकर्ता के अनुकूल इंटरफेस और मजबूत सुरक्षा उपायों के साथ जोड़कर जेएबी कोड प्रौद्योगिकी को व्यापक रूप से अपनाने को बढ़ावा देने की दिशा में एक महत्वपूर्ण कदम का प्रतीक है।

(जलज शर्मा)
सलाहकार

(ऋषभ कुशवाह)
लेखक